GDAŃSK UNIVERSITY OF TECHNOLOGY
Faculty of Electronics,
Telecommunications and Informatics

Zbigniew Paszkiewicz

Recommendation Method RMV for Partner and Service Selection in Virtual Organization Breeding Environments Based on Process Mining Techniques

PhD Dissertation

Supervisor:
prof. Wojciech Cellary
Faculty of Informatics
and Electronic Economy
Poznań University of Economics

Gdańsk, 2014

For Zosia, my parents and my brother Jakub

Acknowledgements

I wish first of all to thank in particular Professor Wojciech Cellary for many intellectually challenging discussions that we have had on the topics related to this dissertation and for keeping me going.

I would also like to thank Willy Picard for his invaluable remarks on this dissertation and his encouraging comments. Without your involvement, this dissertation would have never been written.

I would like to thank the members of the Department of Information Technology of the Poznań University of Economics for their collaboration during these years.

Finally, thanks to my beloved fiancé Zosia for her love, patience, and support. At last, thanks to my parents, brother and my friends because I have never been left alone with my work.

This work has been partially supported by
the Polish National Science Center. Grant no. DEC-2011/01/N/ST6/04205

Table of Contents

1. Introduction

Usually, environment of an organization has significant impact on its success. Current trends: globalization, development and proliferation of information technology, spread of social media, development of electronic, knowledge-based economy and rising competition, are followed by increased complexity, uncertainty, dynamism, turbulence and diversity of organization operations. In such environment, complex production and service provision at the global scale require a large set of resources and competences that one enterprise is usually not able to provide. Thus, modern provision of services and delivery of products require integration and collaboration of many diversified, specialized, autonomous units offering access to complementary set of resources and competences.

As a generic organizational structure supporting collaboration of divers units, the concept of Virtual Organizations (VO) has been coined. VO is "*a network consisting of a variety of actors, called VO members that are largely autonomous, geographically distributed, and heterogeneous in terms of their operating environment, culture, social capital and goals, which conduct processes including at least one VO collaborative process in order to carry out a particular venture due to the demand from VO clients*" (Camarinha-Matos, et al., 2008). VO permits to deal with complexity, pursuit for agility, and takes advantages of broad use of information technologies in economic and managerial operations. Partners collaborating within a VO are organizations – enterprises, public administration units, and non-government organizations – people, and information systems. The success of a VO strongly depends on ability of all participating actors to efficiently and seamlessly collaborate. Good level of collaboration is achieved by appropriate selection of services and collaborators. Due to importance and complexity of partner and service selection problem, a number of computer and organizational methods has already been proposed including the concept of Virtual Organization Breeding Environment (VOBE). A VOBE is *"an association of organizations with the main goal of increasing preparedness of its members towards collaboration in potential virtual organizations"* (Camarinha-Matos, et al., 2008). VOBE allows potential collaborators to prepare their future collaboration with other VOBE members before a business opportunity occurs. A VOBE that consequently applies the Service Oriented Architecture (OASIS Technical Committee, 2006) is referred to as Service-Oriented Virtual Organization Breeding Environment (SOVOBE) (Picard, et al., 2010).

In SOVOBE, interaction among all the actors, virtual organizations and SOVOBE infrastructure is performed with the use of services.

Complexity of organizational environment is followed by complexity of VO collaborative processes guiding operations of VOs. Two main features of each VO collaborative process are: unpredictability and emergence. The *unpredictability* aspect of VO collaborative processes refers to the difficulty to plan in advance the further execution of a VO collaborative process. The *emergence* aspect of VO collaborative processes refers to the influence of VO collaborative process instance execution on itself, i.e., decisions made during VO collaborative process instance execution impact the next activities. As a consequence, VO collaborative processes are highly unstructured.

Proliferation of information technologies and ubiquitous access to the internet via fixed and mobile devices are followed by increased number of processes that are performed by the use of electronic means. The concept of *Process-Aware Information Systems (PAISs)* has been proposed in (Dumas, et al., 2005) to encompass various information systems supporting the lifecycle of any process. A PAIS is defined as *"a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models"* (Dumas, et al., 2005). Examples of PAISs include: production workflow managements systems, ad-hoc workflow systems, and computer supported collaborative work (CSCW) systems. PAISs support for process execution aims at finding a balance between flexibility of process definition and guidance for human actions. Systems that allow greater flexibility offer limited support for user guidance and vice versa. PAISs providing both flexibility and guidance are missing. Moreover, in existing PAISs, guidance is based on predefined process models that become quickly obsolete in dynamic organization environment.

Modern PAISs log enormous amount of data providing detailed information about activities and processes that have been executed. Such data are referred to as *events*. Event logs provide valuable insight into process instance executions. Analysis of event logs may permit to discover factors impacting efficient execution of process instances. Information about those factors may be used to improve efficiency of future process instance executions. Efficient use of information from event logs relies on ability to analyze PAISs' data and draw business-level conclusions concerning process execution success factors.

Discovery of knowledge from large amounts of data is the domain of data mining and machine learning techniques. To capture the notion of process in data mining, the term process mining has been coined. *Process mining* is "*a set of techniques, tools, and methods to discover, monitor and improve real business processes by extracting knowledge from event data available in today's information systems"* (Aalst, 2011). Traditionally, in process mining research, one may distinguish two main areas of interest: on-line and off-line analysis. While off-line analysis includes *process discovery, process conformance checking* and *process enhancement,* on-line analysis encompasses *process prediction, detection* and *recommendation* (Aalst, 2011).

On-line process mining recommendations provide an opportunity to efficiently support processes executions in PAIS (Schonenberg, et al., 2008). The idea consists in analyzing past and ongoing executions of process instances to discover real, actual process models and use them to guide users by PAIS. Recommendation of activities based on event logs is a known but still new and largely unexplored area of process mining.

Existing process mining recommendation methods are limited to structured processes with known process models. Such methods cannot be applied to very large sets of unstructured, emerging, unpredictable processes with unknown underlying process model. Moreover, existing methods permit to discover only activity names and time characteristics. They do not include important aspects relevant for partner and service selection such as information about actors involved in the VO collaborative processes and the context of VO collaborative process executions. The need for efficient process mining methods supporting execution of unstructured processes guiding collaboration is confirmed in (Sinur & Jones, 2012): *"in processes where there are multiple ways of finishing a process instance to completion, typical in semi- and unstructured processes, paths to success can be traced by automated business process discovery. Success patterns can be mined, stored and presented to resources as better practices, ranked by goals for future process participants. This will become more important where collaboration is used to drive to completion. Case management, extreme collaboration and social interactions are drivers for these kinds of process solutions. Productive collaboration pairings and sources can be identified and leveraged for future cases"*.

The main idea of the *Recommendation Method for Virtual Organizations* (the RMV method), proposed in this dissertation, is automatic discovery of activity patterns and ad-hoc generation of recommendations for VO collaborative process instances performed within a SOVOBE. An *activity pattern* is a set of partially ordered activities performed by collaborators that frequently occurs in many instances of VO collaborative processes.

In the RMV method, it is assumed that event logs of a PAIS satisfy the following requirements:

1. Each event in the event log has attributes indicating actors involved in execution of the activity instance comprising the event;
2. Each event has a distinguished set of attributes describing circumstances in which it was recorded;
3. Each completed VO collaborative process instance is described by a set of attributes, including one that indicates the outcome of the VO collaborative process instance.

The RMV method consists of two main phases:

1. Identification of activity patterns and their contexts;
2. Recommendation formulation.

In the first phase, a set of activity patterns is identified. Each activity pattern contains information about its contexts, partially ordered set of activities ordered according to temporal dependencies, involved set of actors, and relations among actors. The second phase is performed on request. In the second phase, activity patterns suited to a particular context of a running VO collaborative process are selected and recommended for inclusion in further execution of the VO collaborative process instance. Once an activity pattern is selected, it is instantiated before incorporation into VO collaborative process execution. Instantiation is based on information stored in the activity pattern regarding actors and information concerning SOVOBE members provided by SOVOBE services. Selection of the best matching activity pattern from the set of recommended activity patterns and its instantiation is performed in a collaborative way by a group of collaborators. Recommendations generated by the RMV method are used by PAIS to guide user actions. As a consequence, PAIS provides a support for both flexible definition of process models and user guidance, where the guidance is based on discovered, real and actual activity patterns.

The thesis of this dissertation is the following:

> *The RMV method permits formulation of accurate recommendations leading to selection of partners and services for collaboration within virtual organizations.*

The reminder of this dissertation is organized as follows. In Section 2, an inter-organization collaboration is characterized in detail. In particular, the theoretical foundations of inter-organization collaboration formulated on the ground of economy and management are presented. This section also explains the application of the Service Oriented Architecture at inter-organization level and formalizes basic concepts referring to collaboration such as: *service*, *process*, *process model*, *collaborator* and *collaboration*. Two important concepts of *Virtual Organization* and *Virtual Organization Breeding Environment* are presented as organizational structures supporting inter-organization collaboration. Finally, the problem of partner and service selection for VO collaborative processes is presented.

Section 3 is devoted to computer support for execution of VO collaborative processes. First, the concept of Process-Aware Information Systems (PAISs) is introduced that support processes modeling, management and adaptation. PAISs are analyzed in terms of support they provide to VO collaborative processes. Next, the concept of service protocols is presented as an approach to modeling and adaptation of collaboration. Finally, context-aware recommender systems are analyzed in terms of on-line support of VO collaborative processes execution. The analysis is divided into two parts: (1) approaches to context modeling, (2) methods of generating recommendations based on modeled context.

Section 4 outlines data analysis methods applicable to human behavior, including those based on process mining. Special attention is paid to the concept of operational support based on process mining and process mining recommendation methods. This section is concluded by description of methods of data analysis, other than process mining, which aim at finding patterns in data describing human behavior.

In Section 5, the concept of the RMV method together with its formal model is presented. First, motivation behind the RMV method is explained in detail. The motivation is based on the requirements of inter-organization collaboration, limitations of existing methods in support of VO collaborative processes, and limitations of current process mining and data analysis methods. The motivation is followed by the RMV method outline which is divided into two parts: (1) general description of the method, and (2) description of the RMV method key aspects. Formalization of the RMV method includes the following concepts: *collaborative process, VO, VO collaborative process context, collaborative process event log, activity patterns, activity pattern recommendation,* and *activity pattern instantiation*, as well as the following operations: *activity sequence pattern discovery, activity pattern identification, recommendation formulation* and *activity pattern instantiation*. The section is concluded by description of the RMV method parameters and the RMV method computational complexity.

Integration of the RMV method with a PAIS is presented in Section 6. First, the architecture of the prototype implementation of the RMV method is presented. Then, the *ErGo* system being an example of PAIS used in the construction sector is presented. The *ErGo* system description includes its main applications: *ErGo Organizations, ErGo Services, ErGo Investments,* and *ErGo Investment Types*. The prototype implementation of the RMV method is integrated with all the *ErGo* applications. Moreover, the *ErGo MatchMaker* application is added to the *ErGo* system as a part of the prototype. This section includes also evaluation of the RMV method applied to real case data. The analysis is performed on data coming from a delivery process performed in the warehouse of Epsilon company which is a medium sized production company. Application of the RMV method has led to discovery of activity patterns and formulation of nontrivial recommendations concerning complaints handling, work of quality department employees, and handling of damaged pallets.

Section 8 concludes the dissertation.

In addition, three appendices are provided. Appendix A contains the formal model of service protocols that are used as a part of formal representation of activity patterns. All the modules and classes of the prototype implementation of the RMV method are presented in Appendix B. Appendix C contains table of symbols used in the dissertation.

2. Inter-Organizational Collaboration

Organizations always perform their processes in a particular economic, legal, social, political and technological environment which has impact on their success (Stoner, et al., 1999). In (Porter, 2008), *organization environment* is defined as *"all the forces, processes and other entities – companies, public administration agencies, non-government organizations, etc. – outside an organization that interact with the organization and can potentially affect the organization's performance"*. Current trends: globalization, development and proliferation of information technology, spread of social media, development of electronic, knowledge-based economy and rising competition, are followed by increased complexity, uncertainty, dynamism, turbulence and diversity of organization operations. Such environment is a particular challenge for small organizations. Although many small organizations are flexible and innovative, so they may adapt to transforming environment in a relatively easy and fast way, they are volatile, they have limited capabilities to influence the market and to control their environment and, finally, they have to compete with large global organizations that have much more resources. To achieve market success, small organizations have to combine strategies of specialization, differentiation and cost leadership throughout the value chain. As a consequence, these strategies lead to collaboration by integration of various organization efforts.

The main reason for collaboration among organizations is the need for *competitive advantage*. The first theoretical framework that may be used to understand the fundamental need for collaboration among organizations has been proposed by David Ricardo in his book *"Principles of Political Economy and Taxation"* (Ricardo, 1817). David Ricardo indicated the strategy of *specialization* as a way to boost efficiency of organization operation. Specialization means concentration of an organization on operations where it has comparative advantage and taking benefits from exchange of goods and services with other specialized organizations. Ricardo has explained that this approach is effective even if an organization is able to produce all the goods and services more efficiently than the other organizations. In the theory of *competitive advantage*, Michael Porter proposed a value chain as an approach to analysis of organization operations (Porter, 2008). A *value chain* is a set of activities related to production processes, marketing, supply, client support, etc. which all together lead to service provision or product delivery that has a value to a final customer. Basing on the value chain concept, two organization strategies were proposed: cost leadership and differentiation. An organization develops a *cost advantage* by reconfiguring its value chain to reduce costs of as many stages of the chain as possible. Reconfiguration means making structural changes, such as adding new production processes, changing distribution channels, or trying a different sales approach. *Differentiation* stems from uniqueness and perceived value. An organization focusing on activities it does best and creating innovative and unique products and services, naturally rises above its competitors. An organization can achieve a differentiation advantage by either changing individual value chain activities to increase uniqueness in the final product or by reconfiguring the entire value chain. Barney (Barney, 1991) has proposed that *"a firm is said to have a competitive advantage when it is implementing a value creating strategy not simultaneously being implemented by any current or potential player"*. Focusing on the resources and attributes which provide the competitive advantage to an organization has a deep impact on its performance outcomes, and therefore, should be a fundamental aspect in every business strategy. As a consequence, organizations should specialize to gain a competitive advantage.

Focusing on competitive advantage leads to narrowing the areas of organization expertise and operation. Meanwhile, the production and service provision currently require a large set of skills and resources that a given organization is usually not able to handle efficiently. Thus, modern value chains cover not one but a number of specialized organizations, integrated with each other to perform activities defined within the value chain. Such collaboration creates an opportunity for efficient, cost effective differentiation at each phase of the value chain. On the other hand, efficient collaboration among autonomous organizations is difficult mainly due to differences among them including geographic, legislative, and cultural differences, diverse markets of products and services, frequent changes of customers and suppliers, as well as modifications of law, technology and methods of work. This raises the question of (1) the methods of modeling collaboration to achieve efficient computer support, (2) organizational structures supporting collaboration. Collaborative processes have been proposed to model inter-organizational collaboration, while Virtual Organizations and Virtual Organization Breeding Environments have been intensively scrutinized organizational structures supporting collaboration among organizations.

2.1. Collaborative Processes

The world economy is currently in an advanced stage of transformation from a goods-based economy to a services-based economy in which value creation, employment, and economic wealth depend more and more on the service sector (Spohrer & Maglio, 2008) (Demirkan, et al., 2008) (Wall, 2007). Service Oriented Architecture (SOA) is one of the widely applied paradigms with relevance to accounting, finance, supply chain management and operations, as well as strategy and marketing. Thus, in modern approaches to management, SOA plays a key role as a way to integrate heterogeneous information systems coming from different organizations (Cellary & Strykowski, 2009) (Picard & Cellary, 2010). Typically, SOA methods, Web services, associated tools and standards enable implementation of SOA at infrastructural and technical levels. Recently, it has been stated that SOA does not have to be limited to these levels only (Picard, et al., 2010). OASIS defines SOA as *"a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains"* (OASIS Technical Committee, 2006). The concept of a service is defined as follows: *"a service is the mechanism by which needs and capabilities are brought together"*. These two definitions emphasize some characteristics of SOA applied to a set of organizations which mutually collaborate to achieve common or compatible goals. The service definition focuses exactly on the matching of the needs of users and the capabilities of an organization or a set of organizations to answer these needs.

The terms traditionally used in SOA can be used to model the collaboration among organizations in the following way. To achieve their goals, organizations perform *activities* defined as closed pieces of work (Workflow Management Coalition, 1999). An activity may be a piece of automated work performed by an information system, e.g., a web service for creating invoices, a piece of work performed by a human, e.g., making a decision by a senior executive, or a piece of work performed by an organization, e.g. constructing a residential building. A set of partially ordered activities which realize an objective in a structured manner is called a *process* (Workflow Management Coalition, 1999). A *process instance* is a single enactment of a process. A *process instance state* is a representation of the internal conditions defining the status of a process instance at a particular moment. A *process model* captures the possibility to execute a given activity in a given state. Information systems, humans and organizations involved in activities being a part of the process are called *actors*. A *service* is an access to a competence of an actor, called *service provider*, to satisfy a need of another actor, called *service consumer*, where the access is provided via a prescribed *interface* (Picard, 2013). A service perceived by a service consumer corresponds to an activity performed by a service provider. A *collaboration* arises when two actors alternately and mutually play roles of service consumer and provider. Actors involved in collaboration are called *collaborators*. A process is *collaborative* if some actors involved in it are collaborators. The formalized definition of a collaborative process is presented in Section 5.4.1.

In the SOA approach, the organizational environment is referred to as a SOA ecosystem. According to OASIS, a *SOA ecosystem* is defined as *"a network of discrete processes and machines that, together with a community of people, creates, uses, and governs specific services as well as external suppliers of resources required by those services"* (OASIS Technical Committee, 2006). Although implementations of SOA at the inter-organizational, infrastructural, and technical levels share many common concerns, such as orchestration of services, reliability issues, and service instance selection, the SOA ecosystem at the inter-organizational level has two specific characteristics: the heterogeneity and dynamism. In the global economy, the SOA ecosystem is neither homogeneous nor static. Following Porter's Five Forces theory (Porter, 1979), SOA ecosystem heterogeneity comes from supplier power, barriers to entry, rivalry among organizations, threat of substitute services and buyer power which are different in different business domains. Also, organizations operating in a SOA ecosystem are characterized by different readiness to personalize provided services, level of computer support in service provision, business models, organizational culture, level of formalization of collaboration and internal processes, geographical localization, ability to adapt to SOA ecosystem changes, etc. Dynamics of a SOA ecosystem follow from constant changes in the set of organizations, individuals and information systems operating in the SOA ecosystem, changes in their operations and their results having impact on the whole ecosystem. In a globalized economic environment, organizations are often competing at the global scale. New organizations are created on a daily-basis, however, for instance in the USA, more than 50% of them do not survive 5 years (U.S. Bureau of Labor Statistics, 2010). Therefore, new potential collaborators appear continuously, while many already collaborating organizations disappear.

2.2. Virtual Organizations and their Breeding Environments

As a generic organizational structure supporting enactment of collaborative processes, the concept of Virtual Organizations (VO) has been coined in (Camarinha-Matos, et al., 2008). In this dissertation, Virtual Organization is defined as follows: "*VO is a network consisting of a variety of actors, called VO members, that are largely autonomous, geographically distributed, and heterogeneous in terms of their operating environment,* *culture, social capital and goals, which conduct processes including at least one collaborative process in order to carry out a particular venture due to the demand from VO clients"*. The formal definition of VO is presented in Section 5.4.1.

Partners collaborating within a VO are organizations – enterprises, public administration units, and non-government organizations – people, and information systems (Cellary, 2006) (Drozdowski, et al., 2005). Theoretical foundations for virtual organizations have been proposed in (Rabelo & Gusmeroli, 2008) and (Camarinha-Matos, et al., 2007). VO permits to deal with complexity, pursuit for agility, and take advantages of broad use of information technologies in economic and managerial operations (Cellary, 2009). In case of a VO, a modification of a VO collaborative process may be triggered either by external events occurring in VO environment or internal events that are strictly connected to the execution of VO collaborative processes. Among others, modification may be followed by change in a set of VO partners, modification of VO collaborative processes or even redefinition of the VO goal. Modification of virtual organizations takes place during the whole VO operation.

Services available in a SOA ecosystem that can be incorporated into VO collaborative processes are offered on markets of services having various forms and levels of formality. For instance, the service market may take form of catalogs of organizations (Panorama Firm, 2012), service auctions (Oferia.pl, 2012), IT service parks (Petrie & Bussler, 2008), dynamic business networks (Bichler & Lin, 2006), Web service ecosystem (Barros, et al., 2005), B2B e-marketplaces (Abramowicz, et al., 2008) or public administration service platforms (Ministry of Administration and Digitization, 2012). Within these markets, service providers offer multiple services that can be dynamically and on-demand bind into VO collaborative processes. Evolution in the area of service publication, discovery and usage recently has shifted from tight coupling of intra-organizational systems to inter-organizational loosely coupling of partners using well defined service level agreements to open market of services with new models of licensing comprising abstract processes and dynamic service selection and instantiation.

A concept of Virtual Organization Breeding Environment (VOBE, sometimes abbreviated to VBE in the literature) has been proposed to support dynamic partner and service selection and instantiation of VO. A VOBE is *"an association of organizations with the main goal of increasing preparedness of its members towards collaboration in potential virtual organizations"* (Camarinha-Matos, et al., 2008). VOBE allows potential collaborators to prepare their future collaboration with other VOBE members before a business opportunity occurs. Preparation is done by publication of available services and provision of other data useful for identification of collaboration chances. VOBE supports its members by providing an access to various services that can be used across virtual organization lifecycle (Camarinha-Matos, et al., 2007):

- In the VO creation phase: VOBE provides access to information not publicly available, such as information about the past performance of VOBE members; it also provides a standardized description of partner profiles, competences and services; it supports the potential partner search and selection; it provides methods and tools for analysis and evaluation of present and future cooperation performance, as well as necessary information for trust building among selected members;
- In the VO operation phase: VOBE supports communication and exchange of documents, facilitates integration of heterogeneous information systems and manages common infrastructure, provides guidelines for standardized data formats, data storage facilities, information about changing environment of collaboration, information about

new collaboration opportunities, and permits to reuse artifacts elaborated by other VOs (in particular business process models, or best practices);

- In the VO evolution phase: VOBE supports adaptation by redefinition of business goals, searching for new partners, supporting negotiations, etc.;
- In the VO dissolution phase: VOBE inherits knowledge, i.e., it captures experience gained during the operation of VOs for future reuse.

While the concept of VOBE is currently widely accepted, many doubts concern its architecture and implementation. Existing VOBEs have been created in an ad hoc manner and have an infrastructure allowing limited support for efficient integration of VOBE members on both business and technical levels. Service-Oriented Virtual Organization Breeding Environments (SOVOBEs) have been proposed as VOBEs organized systematically on both technical and organizational level around the concept of a service allowing SOVOBE members to collaborate better (Picard, et al., 2010).

2.3. Partner Selection for Virtual Organizations

As mentioned in Section 2.2, both SOVOBE and VO operations are based on services performed by organizations, people, and information systems, composed in potentially complex VO collaborative processes. Success of a VO strongly depends on ability of all participating partners to efficiently and seamlessly collaborate. Good level of collaboration may be achieved by an appropriate selection of services and collaborators. Selected collaborators need to be able to mutually communicate, synchronize, and cooperate to efficiently realize a set of activities. Due to importance and complexity of the partner selection problem, a number of computer supported methods has already been proposed. Although, the newest methods address the aspect of service orientation, in the literature the selection of services is usually separated from the concept of selection of partners. While service selection is a concept investigated by the distributed systems community (especially in terms of SOA), partner selection is mainly a subject of interest of the collaborative networked organization community. Nevertheless, in most methods presented in the papers coming from both communities, it is possible to distinguish two elements: first, an *information model* captures and structures information about artifacts based on which selection is performed. Second, various *selection techniques* have been proposed, focusing on the selection context, scope of the selection and selection strategy.

Partner selection is strongly related with the idea of competence modeling (Gallon, et al., 1995). Works on competence modeling aim at providing a structural description of organizations with a special emphasis on competences, profiles, capacities, and resources (Ermilova & Afsarmanesh, 2010) (Pepiot, et al., 2007). The concept of partner selection based on organization and competence profiles have been published in (Ermilova & Afsarmanesh, 2007). In a competence-based approach, inclusion of service characteristics is marginal (Ermilova & Afsarmanesh, 2010) or not present at all (Pepiot, et al., 2007). In (Canfora, et al., 2005) and (Mueller, 2006), the selection of services is based on an information model consisting of service descriptions. A service description usually includes a wide range of technical, functional, non-functional and business characteristics of a given service. In (Claro, et al., 2005) and (Jaeger & Mühl, 2007), opinions concerning services and user feedback are taken into account. These works do not take advantage of the concepts elaborated in the area of competence modeling, so proposed descriptions of service providers are often not structured, consisting of a simple list of attributes. Moreover, while the importance of social aspects in SOA has been noted recently (Świerzowicz & Picard, 2009) (Picard, 2009), existing approaches to inclusion of these aspects in partner selection are still to be developed (Jarimo, 2009). As an example, (Ding, et al., 2003) have proposed a simulation-optimization approach using genetic search for supplier selection, integrating performance estimation, social aspects and genetic algorithm. However, the social relation model encompasses only a simple social model for supply chains limited to only one relation type, i.e. customer-supplier. A number of selection strategies have been proposed for partner selection. The conclusion of the comparison of various popular approaches presented in (Canfora, et al., 2005) and (Crispim & Sousa, 2007) is that genetic algorithms are the most popular approach. The general guidelines for partner selection within the VOBE are presented in (Rabelo & Gusmeroli, 2008) and (Camarinha-Matos, et al., 2007).

2.4. Basic Definitions

In this section, first, the concepts related to objects and classes of objects are defined. Next, the main concepts related to organizations, processes and services in SOA are introduced.

Definition 2.1. (Attribute) An *attribute* a is a pair $\langle an, av_{an}\rangle$, where an is the name of the attribute and av is the value of the attribute. The value of an attribute may be a literal, an attribute or a set of attributes.

Definition 2.2. (Attribute equality) An attribute a_i *is equal* to an attribute a_j if $an_i = an_j$ and $av_{an_i} = av_{an_j}$. Formally, $a_i \bullet a_j = an_i = an_j$ and $av_{an_i} = av_{an_j}$.

Definition 2.3. (Object) An *object ob* is a pair $\langle obn, as\rangle$, where obn is a name of an object and as is a set of attributes $\{a\}$.

Example 2.1. An object named $\boldsymbol{ArchibaldTex}$ is composed of the following set of attributes:

- $\langle \text{Nationality}, Canadian\rangle$,
- $\langle \text{Profession}, \{Architect\}\rangle$,
- $\langle \text{No. realizations}, 17\rangle$. ▪

Definition 2.4. (Object Classifier) An *object classifier* $\underline{ob}$ is a set of attribute names $\underline{ob} = \{an\}$.

Definition 2.5. (Equality According to Classifier) Two objects are *equal according to classifier* $\underline{ob}$ if they have the same attribute values associated with attribute names composing object classifier $\underline{ob}$.

Formally, an object $ob_i = \{obn_i, \{a_i\}\}$ is equal to object $ob_j = \{obn_j, \{a_j\}\}$ according to classifier $\underline{ob}$, denoted $ob_i \overset{ob}{=} ob_j$, iff $\forall\, an_k \in \underline{ob}, \exists \left(a_i = \langle an_i, av_{an_i}\rangle \in ob_i \text{ and } a_j = \langle an_j, av_{an_j}\rangle \in ob_j\right) an_k = an_i = an_j$ and $av_{an_i} = av_{an_j}$.

Example 2.2. All the objects having the same value of attributes named: $Nationality$ and $Profession$ are considered equal according to classifier $ob = \{Nationality, Profession\}$, independently of how many other attributes they have. ▪

Definition 2.6. (Attribute Constraint) An *attribute constraint* a^{α} is a pair $\langle ac, \vartheta_{ac} \rangle$, where ac is the name of the attribute constraint and ϑ_{ac} is a predicate.

Definition 2.7. (Satisfaction of Attribute Constraint) An attribute $a = \langle an, av_{an} \rangle$ satisfies an attribute constraint $a^{\alpha} = \langle ac, \vartheta_{ac} \rangle$, denoted $a \succ a^{\alpha}$, iff $an = ac$ and $\vartheta_{ac}(av_{an}) = true.$

Definition 2.8. (Class) A *class c* is a pair $\langle cn, as^{\alpha} \rangle$, where cn is a name of a class and as^{α} is a set of attribute constraints $as^{\alpha} = \{a^{\alpha}\}$.

Example 2.3. An $ExpiriencedArchitect^{\alpha}$ class is composed of the following set of attribute constraints:
- $\langle \text{Profession}, \supset \{Architect\} \rangle$,
- $\langle No.investments, > 15 \rangle$. ▪

An $ExpiriencedDeveloper^{\alpha}$ class is composed of the following set of attribute constraints:
- $\langle \text{Profession}, \supset \{RealEstateDeveloper\} \rangle$,
- $\langle No.investments, > 10 \rangle$. ▪

Definition 2.9. (Class Instance) An object $ob = \langle obn, a = \langle an, av_{an} \rangle \rangle$ is an *instance of a class* $c = \langle cn, \{a^{\alpha} = \langle ac, \vartheta_{ac} \rangle\} \rangle$, denoted $ob \propto c$, iff $\bigvee a^{\alpha} \in c, \exists a \in ob: a \succ a^{\alpha}$.

Example 2.4. The $ArchibaldTex$ object is an instance of the $ExpiriencedArchitect^{\alpha}$ class, because all the attribute constraints of the class are satisfied: $ArchibaldTex$ is an architect and his number of realizations, equal to 17, is higher than the required number, equal to 15. Note that attribute nationality is not relevant for the $ExpiriencedArchitect^{\alpha}$ class. ▪

Definition 2.10. (Activity) An *activity v* is a closed piece of work aiming at creation of new objects, removal or modification of existing objects.

An activity may be a piece of automated work performed by an information system, e.g., a web service of creating invoices, or a piece of work performed by a human, e.g., making a decision by a senior executive.

Let V^{*} be the set of all the activities.

Definition 2.11. (Activity Description) An *activity description vd* is an object describing an activity with the following minimum set of mandatory attributes having empty values:
- *activityStart* is a timestamp representing the starting time of activity execution, and
- *activityEnd* is a timestamp representing the completion time of activity execution.

Let VD^{*} be the set of all the activity descriptions.

Definition 2.12. (Activity Instance) An *activity instance vi* is a single enactment of an activity.

Let VI^{*} be the set of all the activity instances.

Definition 2.13. (Activity Instance Description) An *activity instance description vid* is an activity description where the values of mandatory attributes are non-empty.

Let VID^{*} be the set of all the activity instance descriptions.

Definition 2.14. (Process) A *process* is a set of partially ordered activities which realize an *objective* in a structured manner.

Definition 2.15. (Process Instance) A *process instance* is a single enactment of a process.

Definition 2.16. (Process Instance Description) A *process instance description pid* is an object describing process instance.

If the process instance is completed, exactly one attribute describes its outcome. Other instance attributes may include information concerning process instance type, purpose, starting time, place of process instance execution, etc.

Definition 2.17. (Process Model) A *process model* π^α is a directed acyclic graph $\pi^\alpha = (VD, E)$, where $VD \subseteq VD^*$ is the set of activity descriptions, and E is a set of arcs determining the partial order of activity execution.

Definition 2.18. (Basic entity) A *basic entity* is a human or an information system.

Definition 2.19. (Organization) An *organization* is a set of at least two members, where a member is either a basic entity or another organization, working within a particular structure of relations in order to achieve a certain goal of this organization and having a plan to achieve this goal. A goal of an organization may change, followed by the need of plan adaptation.

Formally, an organization o_i is a triple $\langle \eta_i, M_i, R_i \rangle$, where:

- η_i is a plan to achieve goal g_i of organization o_i,
- M_i is a set of members of organization o_i, such that $|M_i| \geq 2$,
- R_i set of relations among members of organization o_i, where $R_i \neq \emptyset$.

Let O denote the set of all the organizations $O = \{o\}$.

Definition 2.20. (Actor) An *actor ar* is a basic entity or an organization able to perform an activity.

Let $AR = \{ar\}$ denote the set of all the actors. Note that $O \subseteq AR$.

Definition 2.21. (Actor Description) An *actor description ard* is an object describing an actor.

Definition 2.22. (Competence) A *competence* is an ability of an actor to perform a particular set of activities.

Definition 2.23. (Need) A *need* of an actor is a demand for object creation, modification or removal.

Definition 2.24. (Interface) An *interface* sq is a pair of sets of classes of objects $sq = \langle IN^\alpha, OUT^\alpha \rangle$.

Definition 2.25. (Service) A *service s* is an access to a competence of an actor, called *service provider*, to satisfy a need of another actor, called *service consumer*. The access to a competence is provided via a prescribed *service interface* sq_s, where IN^α is a set of classes of objects accessed by a subset of activities composing service provider competence, and OUT^α is a set of classes of objects to be modified, created or removed during activities execution to satisfy a service customer need.

Example 2.5. Consider two companies: $HuntersComp$ and $Analytix$. Company $HuntersComp$ has a competence of finding highly qualified professionals in the labor market. Finding a right professional requires execution of a set of activities by $HuntersComp$. Company $HuntersComp$ is a provider of a service $EmployeeSearch$. Service $EmployeeSearch$ offers an access to $HuntersComp\ competence$. ▪

The $EmployeeSearch$ service has an interface $sq_{employeeSearch} = \langle IN^{\alpha}, OUT^{\alpha} \rangle$, where IN^{α} is a 1-element set whose element is a class of position description $PositionDescription^{\alpha} \in IN^{\alpha}$ having the following set of attribute constraints: $\{\langle responsibilities, \neq \emptyset \rangle, \langle expirience, \neq \emptyset \rangle\}$. $PositionDescription^{\alpha}$ class states that a position description object must provide a none empty list of responsibilities of an employee and expected experience of an employee. An object being an instance of $PositionDescription^{\alpha}$ class must be provided by a service consumer during the service call. OUT^{α} is a 1-element set whose element is a class of employee bio $Bio^{\alpha} \in OUT^{\alpha}$ having a set of attribute constraints $\{\langle name, \neq \emptyset \rangle, \langle contact\ info, \neq \emptyset \rangle, \langle skills, \neq \emptyset \rangle\}$. The Bio^{α} class states that employee bio with his/her name, contact information and skills will be given to a service consumer as a result of service $EmployeeSearch$ execution.

$Analytix$ is a $HuntersComp$'s client. $Analytix$'s need is to find a proper business analyst. $Analytix$ is a consumer of the service $EmployeeSearch$. $Analytix$ requests service execution and gives a $HuntersComp$ company an object $BusinessAnalyst$ having attributes $\{\langle set\ of\ responsibilities, \{communication\ with\ clients,\ leading\ IT\ projects,$ $managing\ group\ of\ junior\ analysts\} \rangle, \langle experience, 10\ years \rangle\}$. $BusinessAnalyst$ is a description of a business analyst position in $Analytix$. Note that $BusinessAnalyst \propto PositionDescription^{\alpha}$. As a result of service execution, $HuntersComp$ returns object $JohnSnowBio$ with attributes $\{\langle full\ name, John\ Snow \rangle, \langle contact\ info, \text{phone no. } 507098314 \rangle, \langle \text{competences},$ $\{knowledge\ of\ databases, knowledge\ of\ project\ management\} \rangle\}$ to $Analytix$. Note that $JohnSnowBio \propto Resume^{\alpha}$. ▪

3. Computer Support for Collaborative Processes

Efficient computer support for VO collaborative processes requires proper approaches to modeling collaboration, supporting execution of VO collaborative processes and providing guidance to collaborators in determining upcoming process activities that must be performed. The main support is provided by Process-Aware Information Systems (PAISs) and recommender systems.

3.1. Process-Aware Information Systems

3.1.1. Business Process Management

Business Process Management (BPM) is defined as *"supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information"* (Aalst, 2004) (Aalst, et al., 2003). In the literature concerning BPM, information required to model and control a process has been classified according to various perspectives. In (Aalst, et al., 2003), five perspectives have been presented:

- The *functional* perspective focuses on activities to be performed, their casual and temporal dependencies,
- The *process* perspective focuses on the execution conditions for activities,
- The *organization* perspective focuses on the organizational structure of the population that potentially executes activities,
- The *information* perspective focuses on data flow among activities,
- The *operation* perspective focuses on elementary operations performed by applications and resources.

Traditionally, BPM literature describes the following six phases of the process lifecycle (Reichert, 2011):

1. The process *design* phase, during which a process designer or a group of process designers define a process model; the definition of a process model is based on a process modeling language, e.g., BPMN[1], EPC[2], providing appropriate constructs for creating graphical representation of the model;

[1] Business Process Modeling Notation, http://www.bpmn.org/
[2] Event-driven Process Chain, http://www.ariscommunity.com/event-driven-process-chain

2. The process *configuration* phase, during which appropriate implementation of the activities of the process are identified;
3. The process *implementation* phase, during which values of the process parameters, especially values of the parameters of its activities, are provided to tailor the process instance to the addressed situation;
4. The process *enactment* phase, during which activity instances are executed;
5. The process *monitoring* and *diagnosis* phase, during which the execution of the process instance is controlled, logged, and eventually audited;
6. The process *evolution* phase, during which modifications are introduced in the process instance and/or the process model.

Proliferation of information technologies and ubiquitous access to the internet by fixed and mobile devices are followed by increased number of processes that are performed by electronic means. The concept of *Process-Aware Information Systems (PAISs)* has been proposed in (Dumas, et al., 2005) as a concept encompassing various information systems supporting process lifecycles. A PAIS is defined as *"a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models"* (Dumas, et al., 2005). PAISs include among others: workflow management systems, Enterprise Resource Planning (ERP) systems and Customer Relationship Management (CRM) systems. The main difference between more traditional information systems and PAIS is the focus of the system. Information systems take a data-driven approach, focusing purely on the tasks they have to perform. On the contrary, PAISs look at the process they have to support. As a consequence, PAISs are able to support organizations by providing insight in the status of both the process as a whole and activities which are part of it. This enables organizations to monitor and communicate about their current state and performance.

3.1.2. Characteristics of VO Collaborative Processes

Heterogeneity and dynamic nature of the SOA ecosystem is followed by complexity of VO collaborative processes. Collaborators constantly gain knowledge through the analysis of information concerning the organization environment. Moreover, collaborators learn from each other both explicit and tacit rules governing the execution of collaborative processes. As a consequence of instantly gained knowledge, collaborators change the way they perceive processes, activities, semantics of the decisions being made, and the way these decisions have been made. Due to usually long-lasting character of the VO collaborative processes, the set of collaborators and their roles change, so the set of collaborators having the holistic vision and understanding of collaborative processes realized in the whole VO may be small. Finally, similar instances of VO collaborative processes—e.g. having a similar goal, involving a similar set of collaborators, performed at the same time—may be interrelated, which means that the course of execution of one VO process instance and its results may influence the course of execution of another instance.

Traditional PAISs are built on clear separation of design-time and run-time phases. Process design, implementation, and configuration phases are considered to be the design-time phases. Process enactment, monitoring and diagnosis phases are considered to be the run-time phases. Most PAISs enforce a strong precedence constraint on design-time and run-time: a process is first modeled, and then executed, with a limited possibility to change the model at run-time. This strong precedence is usually justified by two assumptions:

1. The process model is known before process execution, i.e., it is possible to design a process model that is further instantiated and later on executed;
2. The business environment is rather static, which implies process repetition.

Processes meeting the above two assumptions are structured, i.e., repeatable and predictable. Examples of such processes are production ones which can be well supported. To support such processes, a number of methods and standards have been proposed in (Russell & Aalst, 2007) and in WS-BPEL (IBM & SAP, 2005), WS-Coordination (OASIS, 2007), WS-Choreography (W3C, 2004) standards. Full automation is achieved at the expense of possibility to provide change in a process model which cannot be made during process run-time.

In case of VO collaborative processes, the above two assumptions are often not observed. This causes the mismatch between a support provided by PAISs and highly dynamic business environments. If a business environment is highly dynamic, it may be impossible to foresee the process that has to be performed at a given moment. Two aspects of VO collaborative processes have to be addressed to tackle their ad-hoc character:

1. The *unpredictability* aspect of VO collaborative processes refers to the difficulty to plan in advance a partially ordered set of activities to reach the assumed goal;
2. The *emergence* aspect of VO collaborative processes refers to the influence of the VO collaborative process instance execution on itself, i.e., decisions made during process instance exaction condition a set of the next activities.

To precisely define problems related to VO collaborative process modifications during run-time, the concepts of process flexibility and adaptation are used (Sadiq, et al., 2005). *Flexibility* refers to the fact that execution of a VO collaborative process starts without its full specification, i.e., the full set of activities to be performed and their ordering is not known when the VO collaborative process execution starts, so specification of the model is made at run-time and may be unique to each VO collaborative process instance. Flexible VO collaborative processes are characterized by a lack of ability to completely predict and define a set of activities and ordering relationships among them. *Adaptability* is the possibility of a VO collaborative process to adjust to exceptional circumstances that may or may not be foreseen, and generally would affect one or a few VO collaborative process instances. As the possibility of modification of VO collaborative process instances at run-time plays a crucial role in a SOA ecosystem, a special attention should be put on computer methods supporting VO collaborative process flexibility and adaptation. Currently, computer support for adaptability and flexibility of processes is provided to various extent in many systems, e.g., YAWL (YAWL Foundation, 2012), ADEPT (Reichert, et al., 2005), DECLARE (Pešic, et al., 2007). A survey of current approaches is provided in (Picard, 2013).

In (Swenson, 2010), a problematic case of processes related with knowledge work has been identified, e.g., emergency rescue, financial audit or bridge construction engineering, for which a new approach is needed, referred to as Adaptive Case Management (ACM). In ACM, it is impossible to predict the full course of process execution. However, it can be noted that some sets of activities are highly probable to appear in particular circumstances. Still, it is unknown whether the particular circumstances will appear. For instance, consider a process describing a rescue action performed by a firefighter. The firefighter does not know how a particular rescue action will develop, but he/she is trained to behave in a certain way in particular circumstances. For example, when an electrical installation is on fire, he/she performs a known set of activities related with this situation. Existence of reproducible, typical human behavior patterns noted in (Swenson, 2010) is supported by findings described in (Anderson, 2009). In cognitive psychology, these typical patterns are referred to as *scripts*. A script describes some specific circumstances and a set of activities that is typical for these circumstances. It has been shown that individuals are able to complete uncompleted or erroneously reported scripts so that errors in the observed situations could be corrected (Anderson, 2009). Similar findings can be observed in various fields. In (Magnusson, 2004), it is stated that *"behavior consists of patterns in time"*. The authors observe typical behaviors in team sport games like soccer. Some works even draws a connection between behavior patterns and patterns found in DNA-sequences (Magnusson, 2005). In (Heierman & Cook, 2003) it is argued that identification of significant patterns in human behavior can boost efficiency of operation of smart home infrastructure. Similar observations are made in the area of interactive user interfaces (Davison & Hirsh, 1998) (Hartmann & Schreiber, 2007). It has been shown that reproducible human behavior patterns concern also groups of individuals while small alterations might exist for individuals coming from different cultures and societies (Anderson, 2009). Processes analyzed by Swenson share characteristics with VO collaborative processes.

The IT support for processes observed by Swenson is not yet provided. To systemize classification of process in accordance with various levels of structuration, in (Aalst, 2011) three levels of process structuration have been proposed:

1. *Structured* processes (also called *Lasagnia-like processes*) – during execution of various process instances more than 80% of activities happen as captured in the process model and stakeholders confirm the validity of the model; structured processes have activities that are repeatable and all the activities have a well-defined input and output;
2. *Unstructured* processes (also called *Spagetti-like* processes) – execution of process instances is driven by experience, intuition, trail-and-error, rules-of-thumb, and vague qualitative information; it is difficult to define pre- and post-conditions for activities;
3. *Semi-structured* processes – it is possible to sketch a general process model and major parts of it are known in detail; conditions under which activities are performed are known and some activities require human judgment; process instances can deviate depending on actors decisions and the specific characteristics of the process instance being executed.

Processes analyzed by Swenson and VO collaborative processes are in between semi- and unstructured processes – it is possible to distinguish repeatable behaviors in generally unpredictable and emerging process instances. In this dissertation such processes will be referred to as *quasi-structured* processes.

3.1.3. Service Protocols

In (Picard, 2013), *service protocols* have been proposed as an approach to modeling VO collaborative processes. The advantages of service protocols over other approaches to VO collaborative process modeling have been listed in (Picard, 2013). In particular, traditional approaches have the following disqualifying features:

- Static set of process actors: assignment of actors and services to the whole process; this assignment cannot be modified at run-time;
- Singular service consumer: in the existing approaches, a single service consumer and multiple service providers are assumed;

- Limited constraints: in the existing approaches, process models focus on the set of activities and their partial ordering; the concept of role is used to limit the execution of a given activity to actors with appropriate rights; the role definition is usually limited to a label associated with a set of activities that may be performed;
- Unsupported social aspects: although in the existing approaches the importance of social aspects in VO collaborative processes has been largely studied, existing methods of process modeling still lack support for relational constraints;
- One-time instantiation: in the existing approaches, the instantiation of a process is done at once for the whole process and this assignment cannot be modified at run-time.

A service protocol proposed in (Picard, 2013) consists of four elements: a process model, a service-oriented summary of a process model, service network and a service network schema. As mentioned in Section 2.4, a process model defines a set of partially ordered activities to be performed during process execution. A *service-oriented summary of a process model* is an association of each activity with a service description, where a *service description* is a triplet defining the “who” (the service consumer), “what” (the service interface), and “whose” (the service provider) part of the activity. A service-oriented summary of a process model provides a representation of the activities of the associated process model in SOA terms, independently of the process modeling language, e.g., BPEL[3] or BPMN. In a service-oriented summary of a process model, each activity of the process is associated with a service represented by a service description. Information about *service entities,* i.e., service providers, service interfaces, and service consumers, are captured in a service network. A *service network* is a directed graph of *service entities*, i.e., service providers, service interfaces, and service consumers. Service network aims at capturing properties and relations among service entities. A service network is the source of service implementation used to instantiate service protocol. A *class of service entities* is set of constraints which service entities being instances of that class must observe. A class of arcs of a service network is a set of constraints that the arcs being instances of that class must observe. Classes of arcs are called *service requirements*. A *service network schema* is a graph composed of classes of service entities and service requirements. A service network schema restricts the set of potential service entities that may participate in a service protocol execution. The constraints should be taken into account when selecting service entities, i.e., actors and service interfaces, during instantiation of the collaborative process model. A service entity is an instance of a class of service entities iff it satisfies all the constraints defined by the class of service entities. Finally, a *service protocol instance* is a service protocol, where activity names and all the classes of service entities defined in service network schema are known. The formal definition of a service protocol is given in Appendix A.

A service protocol may be applied at four levels that differ mainly with regard to the availability of information concerning the chosen service consumers, providers, and interfaces:

1. At the *abstract level*, a service-oriented summary provides a service-oriented representation of a process model, a service network schema provides constraints on service entities and social requirements, and both the service oriented summary and the service network schema are linked to associated service descriptions (from the service-oriented summary) with classes of service entities (from the service network

[3] Business Process Execution Language, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf

schema); graphical representation of an abstract service protocol is presented in Fig. 3.1 (Picard, 2013);
2. At the *prototype level*, service entities of a service network are associated with both service elements of the service-oriented summary and the classes of service entities of the service network schema; at the prototype level, the service network provides only a partial implementation of an abstract service protocol, as some elements of the service-oriented summary and some classes of service entities of the schema may not be associated with any service entity of the service network;
3. At the *executable level*, the service network associated with both the service-oriented summary and the service network schema provides a complete implementation of an abstract service protocol: all the service elements of the service-oriented summary and all the classes of service entities of the schema are associated with service entities of the service network;
4. At the *instance level*, an executable service protocol is enacted; at the instance level, service entities defined at the executable level consume and provide services modifying the state of the process model.

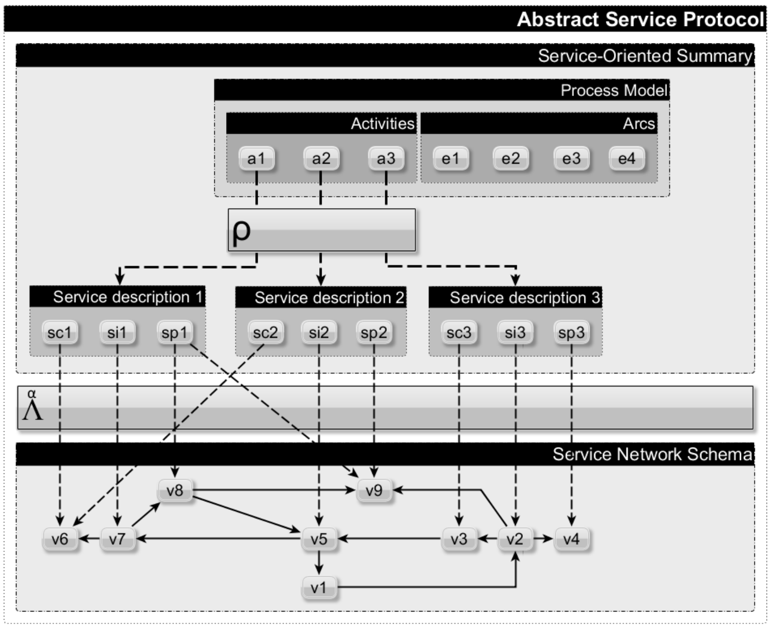


Fig. 3.1. Abstract service protocol

The main characteristics of service protocols that facilitate modeling of VO collaborative processes are:

- *Separation of activities implementation from the process model*: a service protocol include potential interactions among collaborators, however, the interactions are decoupled from implementation of the activities performed by actors; as a consequence, activities of a given process model may be implemented in different ways, using different technologies, or different locations/hosts;
- *Modeling shared responsibility*: in a service protocols, responsibility for execution of activities is shared by different actors; service protocols express responsibility of service consumers for invocation of services as well as responsibility of service providers for the execution of services;
- *Constraints on actors*: service protocols support the definition of constraints on actors, both service providers and consumers; constraints on actors are then used as means to

define the obligations that an actor has to fulfill to participate in a VO collaborative process; constraints on actors concern different aspects of actors such as their competences to perform a given activity;

- *Relational constraints*: due to the importance of social aspects in collaborative processes, social protocols support the definition of relational constraints between actors; relational constraints concern activities that may be performed only by actors with appropriate relations with other actors; computer support for VO collaborative processes should treat relational constraints as an integral part of the model.

Although service protocols capture all the aspects relevant to description of VO collaborative processes, the unpredictable and emerging character of VO collaborative process is only partially addressed. The concept of service protocols is based on the assumption that the full process is modeled before the process execution starts. Optionally, during execution, service protocol is adapted to changing circumstances.

3.2. Context-Aware Recommender Systems

As follows from the research on customer decision making and human collaboration (Lombardi, et al., 2009), behavior of actors involved in execution of VO collaborative process instances largely depends on circumstances in which the VO collaborative process instance execution takes place. For instance, utility of a particular activity to an actor, or attractiveness of potential collaborators to an actor, depend also on other aspects than characteristics of these activities and collaborators. In VO collaborative processes, among others, evaluation of utility or attractiveness of an activity or collaborators to an actor is influenced by the domain knowledge of the actor, time, and current location. It has been proved that decision making, rather than being invariant, is contingent on the circumstances of decision making (Bettman, et al., 1991). Social sciences demonstrate that efficient collaboration is conditioned by social circumstances surrounding the act of collaboration (Picard, 2013).

These observations lead to a conclusion that efficient computer support of VO collaborative processes must include modeling and analysis of circumstances in which a VO collaborative process instance is performed. The approaches to this problem are based on the notion of context and context-aware systems.

3.2.1. Modeling Context of Collaboration

Notion of context has been studied in multiple disciplines. The Merriam-Webster dictionary definition of context is *“the interrelated conditions in which something exists or occurs*” (Merriam-Webster Online, 2013). The Free Dictionary defines context as *“the conditions and circumstances that are relevant to an event, fact, etc.”* (Free Dictionary, 2013). More precise definitions of context come from the following main scientific areas: data mining (Berry & Linoff, 1997), e-commerce personalization (Palmisano, et al., 2008), ubiquitous and mobile context-aware systems (Schilit & Theimer, 1994), databases (Stefanidis, et al., 2007), information retrieval (Jones, et al., 2005), marketing and management (Bettman, et al., 1991), cognitive science and linguistics (Nardi, 1995). Each discipline tends to take its own view that is suited for the particular application area or research. In (Bazire & Brézillon, 2005), 150 different definitions of context were examined from different research and application fields. Examined definitions make evident lack of consensus on context definition. The provided analysis also proves the lack of consensus concerning many essential aspects of a context, e.g., external or internal character of context, static or dynamic nature of context. In the majority of works concerning context, the terms '*user*', '*task*', and '*action*' are used as substitutes of '*activity*', and '*actor*' used in the area of VO collaborative processes, including this dissertation.

When considering information systems, the most commonly cited definition of context is the one formulated by Dey in (Abowd, et al., 1999) in the area of mobile computing and ubiquitous systems: "*context is any information that can be used to characterize the situation of an entity, where an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*". In (Ramakrishnan & Gehrke, 2000), the Dey's definition is discussed and another one is proposed stressing the importance of context for user activity fulfillment: *"context is a set of circumstances surrounding a task that are potentially relevant for its completion"*. The definition is still very abstract and gives no guidelines on the actual representation of context and its use in information system. Nevertheless, this definition is adopted in this section. The formal definition of the context is presented in Section 5.4.2.

All information comprising the context is called *contextual information*. Contextual information is provided by *context sources*. A context source can be a sensor and meter in case of ubiquities computing, an information system, database, person, web site, etc.

The concept of conjunction of context and computing is not new (Jones, et al., 2005). A *context-aware system "adapts accordingly to location of use, the collection of nearby people and objects as well as changes to those objects in time"* (Adomavicius & Tuzhilin, 2008). The definition of context-aware system introduced by Dey in (Dey, 2001) puts an emphasis on a role of such a system in supporting actor's activities: *"context-aware system is a system that uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task".* While this definition addresses only the aspect of information presentation to an actor, Abowd et al. in (Abowd, et al., 1999) refined the specification of context-aware system functionality. Three categories of context-aware system are distinguished on the basis of provided functionality (Abowd, et al., 1999):

- *Presentation of information and actions to a user* – refers to information systems that either present contextual information to the information system user, or use context to propose appropriate selections of actions to the user;
- *Automatic execution of actions* – refers to information systems that trigger a command, or reconfigure itself or other information systems on behalf of the information system user according to changes in contextual information;
- *Tagging of context to information for later retrieval* – refers to information systems that tag captured data with relevant contextual information for future analysis or use.

The use of context in information systems requires its formalization. Various approaches to this problem have been presented in (McCarthy & Buvac, 1994) and (Akman & Surav, 1996). In particular, various predicates expressed in logic-based languages formalize the notion of relations among contexts, clarify concepts of context generalization, context hierarchy, context paradox, situation theory, and operations on context. Among others, the works of *McCarthy*, *Guha*, *Buvac*, *Guinchigia*, *Shoham* were compared in (Akman & Surav, 1996).

The following concepts concerning the context modeling are relevant for computer support of VO collaborative processes and discussed further in this section:

- Representational and interactional views on context;
- Contextual information scope;
- Context abstractions;
- Context dynamics.

As follows from taxonomy provided in (Dourish, 2004), contexts can be classified into two views: representational and interactional. In *representational view,* a context is described by a predefined set of observable attributes, whose structure does not change significantly over time. It is assumed in a representational view that the contextual attributes are identifiable and known a priori and, hence, can be captured and used within the applications. Most of approaches described in literature are based on the representational view on context (Anand & Mobasher, 2007). On the contrary, in the *interactional view* it is assumed that the user's behavior is induced by an underlying context, but the context itself is not necessarily observable. In the interactional view approach there is a bidirectional relationship between activities and underlying context (Bettini, et al., 2010), i.e., context influences activities while different activities give rise to different contexts.

Contextual information describes a certain aspect of context, such as time, location, companion, purpose, etc. In literature, contextual information is referred to as context *features categories*, *types* or *aspects*. The following contextual information was distinguished in (Adomavicius, et al., 2005): user context, physical context, and computing context. Other authors added other contextual information, such as time context, blood pressure, user emotions, focus of attention. The scope of contextual information depends on context-aware system domain. Contextual information can be obtained in three ways (Anand & Mobasher, 2007):

- *Explicitly* – directly approaching relevant people and other sources of contextual information and explicitly gathering this information either by asking direct questions or eliciting this information through other means;
- *Implicitly* – nothing needs to be done in terms of interaction with context-aware system users or other context sources, the source of information is accessed directly and the data are extracted from it;
- *Inferring* – context is derived using statistical or data mining methods; usually to infer contextual information, it is necessary to build a predictive model and train it using appropriate data; examples of used methods include Naïve Bayes classifiers and Bayesian Networks.

There are several approaches to determine the relevance of given contextual information for various uses:

- *Manually* – using domain knowledge of the information system's designer or a market expert in a given application domain;
- *Automatically* – using one of many existing feature selection procedures such as machine learning, data mining, or statistics (Koller & M, 1996) (Liu & Motoda, 1998).

In the representational view on context, contextual information need to be identified and acquired before actual operations based on context analysis are performed. Decisions concerning the scope of relevant and collected contextual information should be done at the application design stage. In (Adomavicius & Tuzhilin, 2008) it is proposed that a wide range of contextual information should be initially selected by the domain experts. Then, after collecting the data, it is possible to apply various types of statistic test to identify which contextual information is truly significant.

A context can be analyzed on several levels of abstractions. For instance, contextual information referring to temperature might hold the value of '24 Celsius degrees' as well as the value 'warm', where such contextual information might originate from identical context sources. The value 'warm' is at a higher level of abstraction than the value of '24 Celsius degrees'. Authors of (Schilit & Theimer, 1994) use the notion of high-level context, low-level context and raw data, to describe various context abstractions levels. *Low-level context* is used synonymously for raw data directly output from context sources. The *high-level context* is contextual information that is processed, i.e., it is aggregated, interpreted, calibrated, cleaned from noise, etc. To provide seamless transition among various levels of context abstractions, the concepts of aggregators and interpreters are proposed in the literature devoted to architectures of context-aware systems. *Aggregator* is a component of a context-aware system collecting multiple, distributed, but logically related pieces of contextual information into a common repository. The need for aggregation comes in part from the distributed nature of contextual information. *Interpreter* is a context-aware system component responsible for rising up the level of abstraction of contextual information. An interpreter typically takes contextual information from one or more context sources or aggregators and produces a new piece of contextual information.

To address the problem of context abstractions, the notion of situation was introduced in (Akman & Surav, 1997). A *situation* is defined as "*a limited portion of the world over some location and time which can be picked out by a cognitive agent*" (Akman & Surav, 1997). In (Dey, 2001), situation is defined as "*a state of context sources, aggregators and interpreters*", where *state* is information captured and stored in some point in time. An approach to capturing situations in context-aware systems using clustering methods is proposed in (Bettini, et al., 2010) and presented in Fig. 3.2 (Bettini, et al., 2010).

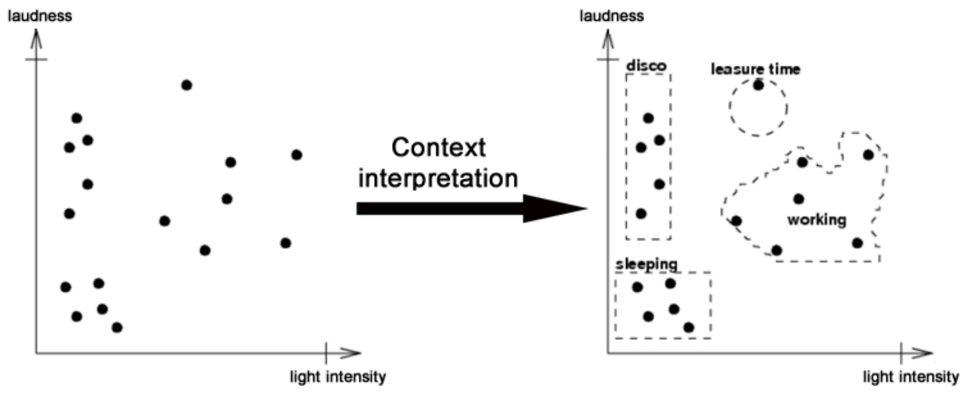


Fig. 3.2. Creating context abstractions

In Fig. 3.2 the context interpretation step is presented using multidimensional coordinate system. The idea is to represent a low level context feature by a vector in a multidimensional coordinate system for every time interval. Each coordinate axis represents a normalized low-level context feature. High level context is then a set of low-level contexts that is assigned a label. A detailed discussion on various aspects of geometrical context representation, such as overlapping of high-level contexts, can be found in (Padovitz, et al., 2004).

In (Adomavicius, et al., 2005), the authors address the notion of context abstractions by building context hierarchies and formulating the concept of context generalization.

In (Adomavicius, et al., 2005) it is proposed that the contextual information is a set of contextual dimensions K, each contextual dimension k in K being defined by a set of q attributes $k = \langle k_1, k_2, \dots, k_q \rangle$. Attributes have a hierarchical structure and capture particular contextual information. The values taken by attribute k_q define finer levels, while k_1 values define coarser levels of contextual knowledge. This is a basis for a formal definition of *context generalization* (Adomavicius, et al., 2005). Let $c' = (c'_1, \dots, c'_k)$, where c_i refers to a contextual information. Then c' is generalization of context $c = (c_1, \dots, c_k)$ iff in the corresponding context hierarchy for every $i = 1, \dots, k$, c'_i is higher in context hierarchy than c_i. Formally, $\forall_{i=1,\dots,k}\ c_i \rightarrow c'_i$. For example, assume three pieces of contextual information, each one described with a hierarchy of possible values:

- *Company*: Girlfriend → Friends → NotAlone → AnyCompany
- *Place*: Theater → AnyPlace;
- *Time*: Sunday → Weekend → AnyTime.

A number of generalizations of context $c = (Girlfirnd,\ Theater, Sunday)$ are possible, among others including the following:

- $c' = (Girlfriend, AnyPlace, Sunday)$;
- $c' = (Friends, Theater, AnyTime)$;
- $c' = (NotAlone, Theater, Weekend)$.

A multiple generalization of one context cause significant computational overload during context analysis. The problem is complex as various levels of granularity have different usefulness in various context-aware systems. For the time being, no generic approach to appropriate context granularity selection was developed.

The dynamic aspect of the context is addressed in research on context prediction (Sigg, 2008). A special emphasis here is put on the notion of time in context analysis. The main assumption is that context is not static but changes over time gradually and semi-predictably (Brown & Jones, 2002) (Greenberg, 2001). The knowledge of context change in history permits extrapolation of context information and prediction of various aspects connected with context. For the purpose of analysis of context change in time, the concept of context time series has been proposed in (Sigg, 2008). A *context time series* are a non-empty, time-ordered set of context elements with attached timestamps, where a *context element* is a non-empty set of contextual information retrieved from context sources during one time interval. In (Brown & Jones, 2002) the concept of context diary was introduced for storage of context information over time.

The concepts of context, context aggregation, interpretation, generalization, etc., have been applied to the area of inter-organizational collaboration quite recently. The application of these concepts in this area is not yet mature. The notion of context appears in publications addressing the problem of virtual organization formation (Tan, et al., 2008) (Do, et al., 2000) (Gonga, et al., 2009). Examples of contextual information relevant to inter-organizational collaboration are (Skopik, et al., 2010): desires, goals and needs of collaborators, ability of service providers to satisfy service consumer requirements, or competences of interacting parties. To enhance inter-organizational collaboration, context analysis is said to be crucial for (Skopik, et al., 2010):

- Determination of collaboration patterns, for example delegation patterns, actor preferences, and actor behavior;
- Selecting suitable collaboration partners and communication channels;
- Analysis of social relations that influence communication patterns.

In (Tan, et al., 2008), seventy eight papers were reviewed to find a common understanding of VO collaborative process context and its relevant scope. The contextual information was divided into seven categories: Role, Activity, Intent, History (Time-based), Social, Emotional/Mental and Others. The contextual information concerning social aspects comes out as very strong and important in (Do, et al., 2000) and (Gonga, et al., 2009). Social aspects of collaboration refer to the nature of interaction between an actor and its human environment. For instance, in (Picard, 2013), the author states that in context of human interactions and inter-organizational interaction, social aspects may limit the choice of collaborators by imposing some relations on interacting entities. In (OASIS Technical Committee, 2006), the concept of *execution context* is proposed as an element of service-oriented architecture. In SOA, the actors exchanging services must agree and acknowledge a consistent set of agreements to successfully collaborate. The *execution context* is *'the collection of consistent set of agreements'*. Furthermore, the execution context concerns *"the totality of the interaction – including the service provider, the service consumer and the common technical infrastructure needed to mediate the interaction"* (OASIS Technical Committee, 2006). The execution context, specific for a VO collaborative process instance, evolves during an execution of the VO collaborative process instance – the set of infrastructure elements, the policies and agreements is changing.

3.2.2. Recommender Systems

As mentioned in Section 3.1.2, VO collaborative processes are knowledge intensive. Observable explosion of the amount of data stored in information systems that is available to actors involved in VO collaborative processes follows the Moore's law[4]. Information concerns other VO collaborative process instances, activities, actors, services, decision points, VO collaborative process instance contexts, etc. One of the main challenges is to extract knowledge from information stored in information systems. The problem of filtering and selection of relevant information on behalf of information system user is addressed by the concept of recommender systems.

Recommender Systems are *"software tools and techniques providing suggestions for items to be of use to a user"* (Ricci, et al., 2011) Usually, recommender systems are targeted to situations when users who lack sufficient experience, competence, time or ability to evaluate overwhelming number of information items potentially suitable for their particular, complex situation, preferences or purposes. Typically, execution of VO collaborative processes follows these characteristics.

Below, evaluation of recommender systems in support of VO collaborative process instances is presented, which encompasses:

- Overview of traditional recommender system elements: item, user model, and transaction logs;
- Study and classifications of traditional two dimensional recommender systems and their algorithms;

[4] *"The number of components in integrated circuits would double every year"*, Gordon E. Moore

- Methods of including context in recommender system, i.e., data models and algorithmic paradigms;
- Limitations of recommender system systems and methods of recommender system evaluation.

recommender systems emerged as an independent research area in the mid-1990s. In recent years, the interest in recommender systems has dramatically increased, mainly due to their important role in highly rated internet sites as Amazon.com[5], YouTube[6], Netflix[7], Tripadvisor[8], Last.fm[9], and IMDb[10]. In operation of every recommender system, three main components are typically distinguished: *items*, *user model*, and *recommendation transaction log*. Recommendations relate to various decision-making processes, such as which products to buy, which music to listen to, which on-line news to read, or which activities to perform.

An *item* is a general term used to denote what the system recommends to its user. Examples of items are: activity, restaurant, article, or movie. Typically, a recommender system is tailored to provide suggestions for a specific type of items. In the most common case, recommendations are presented as ranked lists of items. To rank items, recommender systems try to predict suitability of items for a user, basing on his/her preferences and constraints. To complete such a computational task, recommender systems collect user's preferences, which are either explicitly expressed, or are inferred by interpreting user actions. Items may be characterized by their *value*, *cost* and *complexity* (Ricci, et al., 2011). The *value* of an item may be positive if the item is useful for the user, or negative if the item is not appropriate and leads to wrong decision when selecting it. When a user is acquiring an item, he/she will always incur *cost*, which includes: (1) the cognitive cost of searching for an item, (2) the real monetary cost eventually paid for the item. If a selected item is relevant to a user, this cost is dominated by the positive benefit of having acquired useful information. If the item is not relevant, the net value of a recommendation is negative. *Complexity* of item varies from low (books, videos) to high (cars, insurance, policies, jobs). Simple items can be repressed by name or identifier, while highly complex ones can be represented by developed structures of semantically annotated attributes. More complex is an item, higher is potential benefit of recommendations, but also the difficulty of issuing a recommendation and possible consequences of wrong recommendations. It is up to recommender system developer to design recommender system mechanism to be suitable for a particular type of item considering its value, cost and complexity characteristics.

Provision of recommendations is not limited to human individuals. Recommendations are also formulated for groups, organizations and other information systems. In general, an entity receiving recommendations from a recommender system is called *recommendation system user*. The *user model* encodes recommender system user preferences and needs (Jannach, et al., 2010). As literature focuses on recommendation systems devoted to human individuals, many human user models have been developed. In (Fisher, 2001) the user model is extended by information about relations existing among users, e.g. relations of trust, while in (Taghipour, et al., 2007) users can also be described by data concerning their behavior.

[5] Amazon, http://www.amazon.com/
[6] YouTube, http://www.youtube.com/
[7] Netflix, http://www.netflix.com/
[8] Tripadvisor, http://pl.tripadvisor.com/
[9] Last.fm, http://www.last.fm/
[10] Internet Movie Database, IMDb, http://www.imdb.com/

A *recommendation transaction log* is a *"recorded interaction between a user and the recommender system"* (Ricci, et al., 2011). These recorded data are useful for the recommendation generation algorithm that the recommender system is equipped with. A recommendation transaction log contains a reference to the item selected by the user and a description of the context (e.g., the user goal/query) for that particular recommendation. If available, a recommendation transaction log also includes an explicit feedback the user has provided, such as the rating of the selected item. In refined recommender systems, such data collected during transactions is used for recommender system active learning aiming to refine the recommendation generation algorithm (Ricci, et al., 2011).

Formally, a recommender system rating function is $R\colon User \times Item \rightarrow Rating$, where *User* and *Item* are the domains of users and items, respectively, and *Rating* is a set of items totally ordered according to item utility to a recommender system user (Fig. 3.3) (Adomavicius & Tuzhilin, 2008).
The rating function R is usually defined as a function, where the initial set of ratings is known. Once the function R is calculated for the whole two-dimensional $User \times Item$ space, a recommender system recommends k highest-rated items for each user (*cf.* Fig. 3.3). Some recommender systems do not fully calculate the utility before making a recommendation, but they apply some heuristics to hypothesize that an item is useful to a user (Burke, 2007).

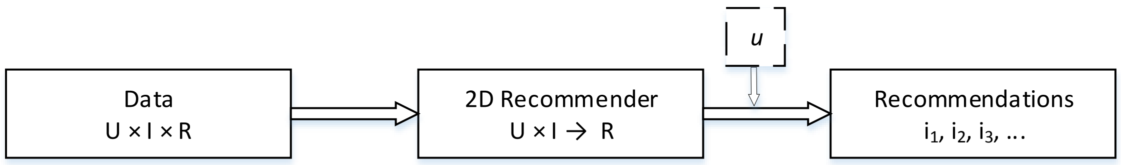


Fig. 3.3. General components of the traditional recommendation process: data (input), two-dimensional recommender system (function), and recommendation list (output)

Assuming that there is some available knowledge (zero knowledge is admissible) about a user who is requesting a recommendation, knowledge about items, and other users who received recommendations, the system will leverage this knowledge with an appropriate algorithm to generate various utility predictions and hence recommendations. Different types of recommender systems can be distinguished that vary in terms of used knowledge, recommendation algorithm, final assemble of recommendations, forms of presentation of recommendations to the user. Coherent classification of recommendation systems is presented in (Burke, 2002). Six different classes of recommendation systems are:

1. *Content-based* (Pazzani & Billsus, 2007) – recommender system recommends items that are similar to the ones that the user liked in the past, where the similarity of items is calculated based on the features associated with the compared items; content-based approach to recommendation has its roots in information retrieval and information filtering research (Abramowicz, 2008);
2. *Collaborative filtering* (Schafer, et al., 2007) – recommender system recommends the items to the user that other users with similar taste liked in the past, where the similarity of two users is calculated based on the similarity in their rating histories;
3. *Demographic* (Mahmood & Ricci, 2007) – recommender system recommends items based on the demographic profile of the user;
4. *Knowledge-based* (Bridge, et al., 2006) – recommender system recommends items based on specific domain knowledge about how certain item features meet recommender system users' needs, preferences and requirements; two types of recommender system are distinguished in this category: (1) case-based recommenders

determine recommendations on the basis of similarity metrics, (2) constraint-based recommenders exploit predefined knowledge bases that contain explicit rules about how to relate customer requirements with items;

5. *Community-based* (Arazy, et al., 2009) (Golbeckm, 2006) (Groh & Ehmig, 2007) – recommender system recommends items based on the preferences of the user friends; in some cases social-network data yields better recommendations than user model profile similarity; adding social network data to traditional collaborative filtering improves recommendation results; this approach is often referred to as *social filtering*; the approach is currently intensively exploited by companies such as Google;
6. *Hybrid recommender systems* (Burke, 2007) – recommender system recommends items based on combination of the other techniques; a hybrid system combining techniques A and B tries to use the advantages of A to fix the disadvantages of B.

Hybrid recommender systems are those that combine two or more of the techniques described above to improve recommendation performance, usually to deal with the cold-start problem of handling new items or new users. Different types of hybrids has been identified and presented in (Burke, 2002).

Each of recommender system, despite the class it belongs to, uses some kind of an algorithm to make the rating function total. To this end, hundreds of algorithms have been developed. In general, used algorithms are divided into two groups:

- *Memory-based* – ranking predictions are based on the entire collection of items previously rated by the users; statistical methods are used that work better as the available set of data gets bigger;
- *Model-based* – collection of ratings is used to learn the predictive probabilistic model (i.e. decision trees, regression); the created model is later used for prediction of unrated items for the user.

A description and analysis of various memory-based and model-based algorithms can be found in (Adomavicius & Tuzhilin, 2005), (Herlocker, et al., 2004), (Popescul, et al., 2001), (Sarwar, et al., 2001). In Tab. 3.1 (Adomavicius & Tuzhilin, 2005) only the most commonly used techniques and algorithms in context-based, collaborative and hybrid systems are enumerated.

Tab. 3.1. Techniques and algorithms used in various classes of recommender systems

Recommendation Approach	**Recommendation Technique**	
	Memory-based	**Model-based**
Content-based	• TF-IDF (information retrieval) • Clustering	• Bayesian classifiers • Clustering • Decision trees • Artificial neural networks
Collaborative	• Nearest neighbor (cosine, correlation) • Clustering • Graph theory	• Bayesian classifiers • Clustering • Artificial neural networks • Linear regression • Probabilistic models
Hybrid	• Linear combination of predicted ratings • Various voting schemes • Incorporating one component as a part of heuristic for the other	• Incorporating one component as a part of model for the other • Building one unifying model

3.2.3. Context-Based Recommendations

Vast majority of the existing approaches to recommendation do not take into consideration any contextual information. Traditional recommender systems do not put users and items into a context when providing recommendations. In VO collaborative processes, evaluation of utility of an activity or a collaborator to an actor is strongly influenced by context (*cf.* Section 3.2.1).

Recommendation systems that use contextual information to improve accuracy of recommendations are called *Context-Aware Recommendation Systems (CARS)*. CARS deal with modeling and predicting user tastes and preferences by incorporating available contextual information into the recommendation processes. CARS are usually modeled as the function $R: User \times Item \times Context \rightarrow Rating$, where *Context* specifies the contextual information.

The approach presented in (Herlocker & Konstan, 2001) serves as a successful illustration of how additional relevant information can be incorporated into the standard collaborative filtering approach. The importance of including and using contextual information in recommendation systems has been demonstrated also in (Adomavicius, et al., 2005), where authors present a multidimensional approach to recommendations. In (Oku, et al., 2006) it is empirically proved that context-aware approach significantly outperforms the corresponding non-contextual approach in terms of recommendation accuracy and user's satisfaction with recommendations.

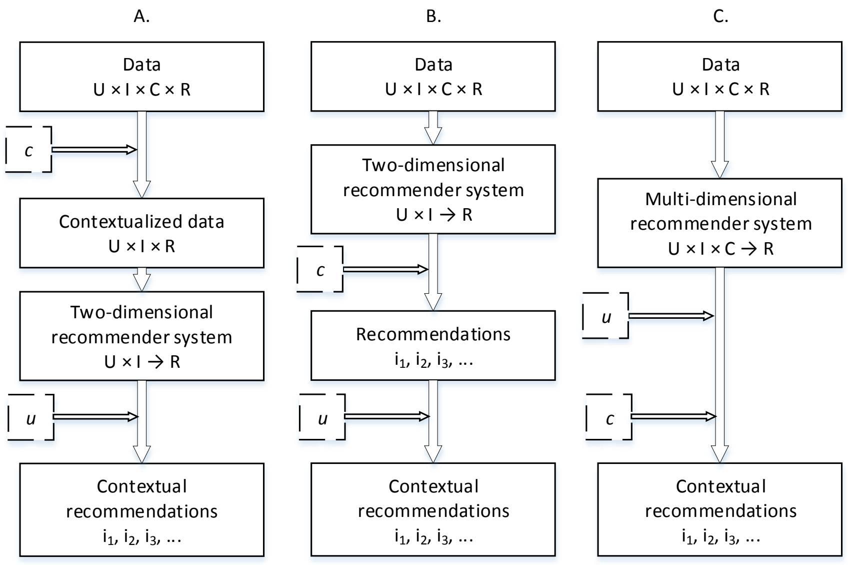
Different approaches to contextual information application in recommendation processes are classified into two groups (Abowd, et al., 1997):

1. Recommendation via context-driven querying and search;
2. Recommendation via contextual preference elicitation and estimation.

The context *querying and search approach* has been used by a wide variety of mobile and tourist recommender systems (Abowd, et al., 1997) (Van Setten, et al., 2004) (Carolis, et al., 2009). Recommender systems based on this approach typically use contextual information to query or search a certain repository of items and present the best matching resources. For instance, a restaurant recommender system may recommend best matching restaurant basing on user's current mood or interest provided explicitly, and additional information collected implicitly describing the user's environment, e.g., local time, weather, or current location. User mood, interest, time, weather, and location form the context of recommendation. The approach to recommendation based on *contextual preference elicitation and estimation* is more recent trend in research, and practical applications (Panniello, et al., 2009) (Yu, et al., 2006). This set of techniques attempt to model and learn user preferences, e.g., by observing the interactions of this and other users with the systems or by obtaining preference feedback from the user on various previously recommended items. To model users' context-sensitive preferences and generate recommendations, these techniques typically either adopt existing collaborative filtering, content-based or hybrid recommendation methods to context-aware recommendations, or apply various intelligent data analysis techniques. Some practical examples of CARS, combine the techniques from both general approaches into a single system. The UbiquiTO system (Cena, et al., 2006) implements a mobile tourist guide and provides intelligent adaptation based not only on the specific contextual information, but also on various rule-based and fuzzy set techniques to adapt the application content based on user preferences and interest. The News@hand system (Cantador & Castells, 2009) uses semantic technology to provide personalized news recommendation that are retrieved using user's concept-based queries or calculated according to a specific user's or user group's profile.

Three different algorithmic paradigms for incorporating contextual information into the recommendation process are discussed:

- *Pre-filtering* (contextualization of recommendation input, Fig. 3.4a) – information about the current context is used for selecting or constructing the relevant set of data records; only the information that matches the current usage context is used to compute the recommendations, e.g., the ratings for items evaluated in the same context; recommendation can be performed using any traditional two-dimensional recommender system on the selected data;
- *Post filtering* (contextualization of recommendation output, Fig. 3.4b) – initially the recommendation algorithm ignores the context information; the ratings are predicted using any traditional two-dimensional recommender system on the entire data; the output of the algorithm is then filtered to include only the recommendations that are relevant in the analyzed context;
- *Contextual modeling* (contextualization of recommendation function, Fig. 3.4c) – context data are explicitly used in the prediction algorithm.



A.
B.
C.
Data U × I × C × R
c
Contextualized data U × I × R
Two-dimensional recommender system U × I → R
u
Contextual recommendations i1, i2, i3, ...
Data U × I × C × R
Two-dimensional recommender system U × I → R
c
Recommendations i1, i2, i3, ...
u
Contextual recommendations i1, i2, i3, ...
Data U × I × C × R
Multi-dimensional recommender system U × I × C → R
u
c
Contextual recommendations i1, i2, i3, ...

Fig. 3.4. Paradigms for incorporating context in recommender systems

In (Adomavicius, et al., 2005), the authors refer to *pre-filtering* approach as *reduction based*. In practice, the use of context in pre-filtering approach can be perceived as formulation of a query for selecting (filtering) relevant data to be used in further analysis. As an example, if a recommendation should concern analysis of activities to be performed by a user on Sunday, only the historical information concerning Sunday activities should be used. The example presents *exact pre-filtering*. Also other pre-filters can be distinguished like *generalized pre-filer*. Following the example, below the exact contextual pre-filer for activity recommendation taking into account time is defined:

$$\forall(u,a,t) \in U \times A \times T, (u,a,t) = R^{D}_{User\times Activity\times Time}(u,a,Sunday) \\ = R^{D[Time=Sunday](User,Activity,Rating)}_{User\times Activity}(u,a),$$

where *User*, *Activity* and *Time* are the domains of users, activities and time, respectively. D is a set of existing ratings. D contains records $\langle user, activity, time, rating \rangle$ for each known, user-specified ratings. Moreover, [Time = Sunday] denotes simple contextual pre-filter and $D[Time = Sunday](User, Activity, Rating)$ denotes a rating dataset obtained from D by selecting only the records where *Time* dimension has value *Sunday* and keeping only the values from *User* and *Activity* dimensions.

Using the concept of hierarchy in context modeling (*cf.* Section 3.2.1), the concept of *generalized pre-filtering* was introduced in (Adomavicius & Tuzhilin, 2008), where simple filter [Time = Sunday], which represents the exact context of the rating *(u, a, t)*, can be replaced by generalized filter $[Time \in S_t]$, where S_t denotes some superset of context t. The three-dimensional reduction can be generalized to reduction of n-dimensional space to m-dimensional one, where $m < n$. To use reduction outputs in recommendation, frequently in practice $m = 2$ is used. Due to the usual multiplicity of possible generalization of one context (*cf.* Section 3.2.1), the important problem arises of selecting appropriate generalized filter. Such selection can be performed based on expert knowledge or automatic selection methods which still need to be developed or refined. When using the pre-filtering approach, one must be aware of the tradeoff between having more relevant data for calculating an unknown rating based only on ratings with the same or similar context and having fewer data points used for this calculation (sparsity problem). This tradeoff leads to better recommendation efficiency of pre-filtering approach in some domains and the general approach in others.

Contextual *post-filtering* ignores context information input data when generating recommendations (Fig. 3.5) (Schilt, et al., 1994). Instead, when a recommendation list is formulated, it is adjusted to user context. The idea behind adjustment of recommendation list goes beyond simple filtering. The approaches based on heuristics and models consists of analyzing contextual preference data for a given user in a given context to find specific item usage patterns and then use these patterns (Schilt, et al., 1994).

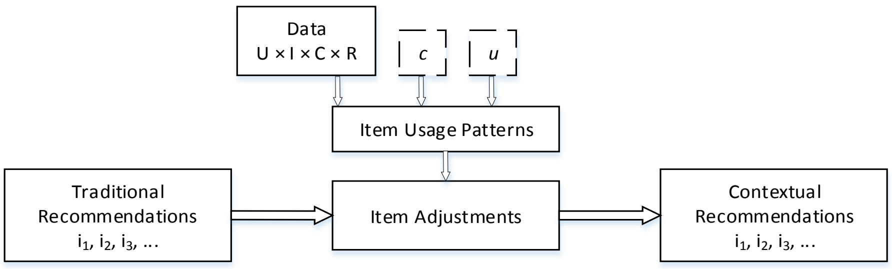


Fig. 3.5. Adjustment of recommendations in post-filtering

Heuristic post-filtering approaches focus on finding common item characteristics (attributes) for a given user in a given context and use these attributes to filter out the recommendations without having less than assumed number of common attributes. Alternatively, items are ranked according to the number of supported attributes. Model-based approach to pre-filtering also relies on filtering out or ranking items in a recommendation list, but such adjustments are based on probability with which the user chooses a certain type of item in a given context. The big advantage of both pre- and post-filtering approaches is that they do not require development of any new recommendation techniques. Instead, traditional techniques developed for two-dimensional recommender system can be used.

Contextual modeling approach uses contextual information in the recommendation function as an explicit predictor of a user's rating for an item. This approach takes advantage of such techniques as decision trees, regression, probabilistic models, etc., or heuristic calculations that incorporate contextual information. Comparison of various methods based on pre-filtering, post-filtering and context modeling is presented in (Panniello, et al., 2009). The comparison points out that the usefulness and efficiency of recommendations based on these methods largely depends on a given application.

The concept of recommender systems is quite mature. Thus, the number of limitations and challenges that the recommender systems must face is well defined. As the recommender systems are specific for particular areas, the impact of various limitations on their performance varies. The universal metrics have been proposed to evaluate and compare recommender systems. Traditionally, four limitations of recommender systems are mentioned in literature: (1) cold start problem, (2) sparsity problem, (3) limited content analysis, and (4) overspecialization.

A *cold start problem* appears when a system has insufficient data concerning users or items that are new in the system. In such situation, the system has no data to use in the recommendation algorithm. This problem can be divided into two types: new user problem, and new item problem. The *new user problem* refers to situation when not enough information is available to build user model. To deal with this problem, it is essential to select training points prior to any recommendation, to be rated by the user that will maximize understanding of what the new user wants. A *new item problem* appears in collaborative filtering approach to recommendation when recommendation mechanism would fail to consider items which no-one in the community has rated previously. The cold start problem is often reduced by adopting a hybrid approach between content-based matching and collaborative filtering. New items would then be assigned a rating automatically, based on the ratings assigned by the community to other similar items according to the items' content-based characteristics. In some approaches to collaborative filtering it is assumed that new products which are inserted into the system are submitted to selected users for evaluation which should result in quick improvement of prediction accuracy.

The *sparsity problem* comes from the observation that usually the number of items stored in recommender system is extremely large. Even the most active users will only rate a small subset of them, so even the most popular items have very few ratings. Thus, too few pairs of users or items have sufficient number of ratings to form a similar group among them. A limited content analysis appears when there is an insufficient amount of available information describing items, i.e., small set of features can be analyzed in terms of similarity among items or users.

Finally, the *overspecialization* problem leads to recommendation of too similar items. Initially, the overspecialization problem concerned similarity in terms of item features. When social filters started to be applied, overspecialization also concerns social relations – typically people being in users' social network like similar items (Pariser, 2011).

The challenges that are important when considering CARS to support VO collaborative processes are (Ricci, et al., 2011) (Adomavicius & Tuzhilin, 2005) the following:

- *Explanations formulation* – challenge refers to formulation of user understandable explanations accompanying recommendations justifying particular recommendations; refined explanation mechanism should explain how the system works, allow users to tell the system it is wrong, increase user confidence in the system, help users to make good decisions, convince users to follow a recommendation, help users to make decisions faster;

- *Exploration versus exploitation* – challenge refers to the dilemma whether to keep items that the system can now identify as good recommendations, given the data currently available for the system or to further explore and complement user preferences to build newer and possibly better recommendations in the future;
- *Time value* – challenge refers to the fact that a given set of recommendations may not be applicable forever but there could be a time interval when these items should be recommended;
- *User activity interpretation* – challenge refers to analysis of many activities performed by the user operating the recommendation system that can be detected, analyzed and used to build a better prediction model;
- *Scalability* – challenge refers to the ability of CARS algorithms to deal with large and real-world datasets;
- *Pro-activeness* – challenge refers to functionality of CARS to provide recommendations even if not explicitly requested by a user;
- *Conversational CARSs* – a system may also request additional user preferences to provide the user with better results; in the transaction model, the system collects various requests-responses, and may eventually learn to modify its interaction strategy by observing the outcome of the recommendation process;
- *Context usage* – challenge refers to efficient use of contextual information in practice;
- *Active learning* – challenge refers to applying active learning methods in CARSs to learn more about user preferences to improve personalization of the recommending process;
- *Optimum feature selection* – challenge refers to efficient identification of relevant features of users, items, and social relations that are used in recommendation processes; challenge refers also to selection of appropriate information describing context of recommendations, as well as to selection of the right context level of abstraction and contextual filter used in multi-dimensional CARSs.

Several metrics have been developed to evaluate efficiency of a particular recommendation system. Metrics allow various CARSs to be compared and problems in CARS implementations to be detected. Typically the following metrics are used:

- *Accuracy* – measure of the differences between values predicted by a CARS and the values actually observed; a root-mean-square error is a frequently used measure of accuracy;
- *Diversity* – measure indicating the diversity among items recommended to a user; the metric is used to diagnose possible overspecialization problem;
- *Coverage* – measure of the percentage of requests for recommendation, a recommender system is capable to make predictions;
- *Usefulness* – measure of user satisfaction from receiving a recommendation;
- *Novelty* – measure of the percentage of times an item recommended to a user has not been recommended before; the metric is opposite to measuring the percentage of times an item already known to a user is recommended.

4. Process Mining

The term *process mining* is used to describe *"techniques, tools, and methods to discover, monitor and improve real business processes by extracting knowledge from event logs commonly available in today's information systems"* (Aalst, 2011). The concept of process mining is based on observation that creating a process model, as stated in the classical business process management approach, is complicated and time-consuming. Moreover, typically there are discrepancies between the actually executed processes and the envisioned process models. Even more, ad-hoc processes cannot be modeled due to their unpredictability and emergence, but once they are executed, the knowledge concerning the model of executed process is still useful. Process mining is based on exploration of events generated by PAIS during the execution of process instances. Exploration aims at discovering process models describing actually executed process instances and facts associated with those models.

Modern PAISs log enormous numbers of events providing detailed information about the activities that have been executed. A finite ordered sequence of events recorded for a particular process instance is called a *trace*. *Event log* is a set of traces. The following assumptions concerning events recorded in event logs must be satisfied: (1) each event refers to one process activity instance, and (2) events are totally ordered, i.e., in a log, events are recorded sequentially even though tasks may be executed in parallel. The formal definitions of event, trace and event log are presented in Section 4.5. The role of event log in process lifecycle is presented in Fig. 4.1. While design phases are based on models, run-time phases are organized around the event log. Enactment of a process inserts data in event log. Process mining can be used in monitoring, diagnosis and evolution phases.

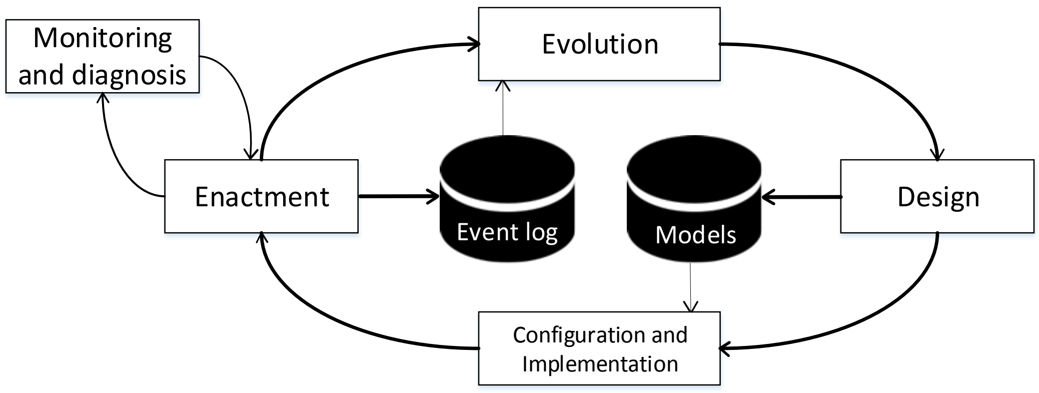


Fig. 4.1. Event log creation and analysis in process lifecycle

Traditionally, in the process mining research, one may distinguish two main areas of interest: *on-line analysis* also referred to as *operational support*, and *off-line* analysis. *On-line* analysis methods encompass process prediction, checking and recommendation. They are described in detail in Section 4.1. *Off-line* analysis encompasses three groups of methods (Aalst, 2011):

- *Process discovery* – methods aiming at discovering process models describing behavior recorded in event log; during a discovery no assumptions are made concerning the resulting process model (Weijters & Aalst, 2001) (Aalst, et al., 2009);
- *Process conformance checking* – methods aiming at comparison of envisioned process model with process instances recorded in an event; conformance checking is used to check if process instances, whose execution is recorded in the log, conforms to the model and vice versa (Aalst, 2011) (Rozinat & Aalst, 2008);
- *Process enhancement* – methods aiming at extension or improvement of an existing process model using information about the process instances recorded in an event log (Hornix, 2007).

Process mining covers different perspectives. The four most popular perspectives include (Aalst, 2011):

1. *Control-flow perspective* focuses on ordering of activities;
2. *Organizational perspective* focuses on information about resources hidden in the event log to classify people in terms of roles and organizational units or to show the social network;
3. *Case perspective* focuses on properties of cases;
4. *Time perspective* is concerned with timing and frequency of events.

While majority of methods concentrate on one individual perspective, some methods aim at finding correlations among various perspectives. This is referred to as *multi-perspective process mining*.

Two types of process models and two types of data are distinguished and used in process mining methods. A *de jure* process model is normative, i.e., it specifies how things should be done or handled. For example, a process model used to configure a BPM system is normative and forces people to work in a particular way (Aalst, 2011). A *de facto* model is descriptive. Its goal is not to steer or control reality. Instead, *de facto* models aim at capturing reality. *Post-mortem* event data refers to information about process instances that have completed. Such data are used in *off-line* process mining. *Pre-mortem* event data refers to process instances that are still going and have not yet completed. Such data are used to support the process instance it refers to by validation of its enactment, prediction of further flow and guidance in execution by provision of recommendations. Analysis of such data is performed in *on-line* process mining. The prototypes of some off-line and on-line methods have been implemented as a part of open-source process mining tool developed by academic community called ProM (Aalst, et al., 2009).

4.1. Operational Support

Process mining techniques can be used to extract knowledge from event logs helping organizations to gain insights into the operation of their business processes. When such analysis is provided for information concerning ongoing process instances in an on-line setting, it is commonly referred to as *operational support*. In (Aalst, 2009), the author outlines the potential of operational support based on process mining. The methods for process prediction, process checking and recommendation have been proposed.

On the contrary to traditional process mining techniques, operational support considers pre-mortem event data. Operational support techniques analyze pre-mortem data and react to them in on-line manner. As running process instances are considered, the concept of *partial trace* is introduced as a trace corresponding to a process instance that is still running. The partial trace describes the known past of the case, while the future of the case is not yet known. Information stored in a partial trace supplemented by information concerning underlying process model and traces of other similar process instances can be used to reason about the ongoing process instance. To this end, three groups of methods have been proposed in (Aalst, 2011):

- *Detection* methods – comparison of *de jure* models with *pre-mortem* data with the goal to detect deviations at run-time;
- *Prediction* methods – combining information about running process instance stored in partial trace with discovered or hand-made process models to make predictions about the future, e.g., the remaining flow time, the probability of success;
- *Recommendation* methods – combining information about running process instances stored in a partial trace with discovered or hand-made process models to suggest to actors suitable activities to be performed to meet objectives of predefined target function, e.g., to minimize cost or time.

Detection has a lot in common with conformance checking but here an immediate response is provided by a system when the deviation occurs. This response should take a form of an action occurring in process-aware system or at least notification displayed by detection mechanism to the user. Two approaches to detection of deviations at run-time are presented in (Aalst, 2011): approach based on workflow net (WF-net) replay and approach based on validation of LTL rules.

In the approach based on WF-net replay, existence of a WF-net is assumed describing the desired normative behavior. Execution of every activity and resulting persistence of event data are followed by verification of a particular trace against underlying WF-net. As far as a partial trace can be replayed by WF-net, the process instances are perceived as to be executed as envisioned. As soon as replaying an event log is not possible by WF-net, the alert signals about the nature of deviation are issued, e.g., about some activity that was started without being enabled.

The approach based on validation of LTL rules emerged from the works on DECLARE workflow system (Montali, et al., 2010) (Aalst, et al., 2009). Declare is a constraint-based workflow system that guides the execution of process instances by constraints that are imposed on these instances. On the contrary to workflow procedural languages that aim at defining control-flow perspective of the process specifying a set of possible actions, in DECLARE *“everything is possible unless explicitly forbidden”* (Montali, et al., 2010). Constraints in DECLARE use semantics based on LTL (Aalst, et al., 2005). The same semantics is used in on-line detection. An example of an LTL constraint is: eventually activity A is executed then activity B and C are executed. The constraint is represented formally as $<>\,((\,activity\ ==\ A \wedge <>\,((\,activity\ ==\ B \wedge <>\,(\,activity\ ==\ C\,)\,)\,)\,)$.
In general, a partial trace is violated if at least one of the constraints imposed on it is violated. In such case, a deviation is detected and reported. A partial trace meets constraints if all the constraints imposed on it are satisfied.

The concept of *prediction* is examined in (Aalst, 2011). Predictions are generated on the basis of information stored in a partial trace and a predictive model. Supervised learning techniques are used as a predictive model, e.g., regression analysis, or decision tree learning. In these approaches, events comprising a partial trace and their relevant properties are mapped onto predictor variables. The response variable is typically a key performance indicator or a probability measure. Examples of predictions include: remaining flow time of process instance, probability of meeting a legal deadline, total cost of process instance, probability of occurrence of a particular activity. The predictive model is based on post-mortem data. It is used for predictions of running process instances.

In (Aalst, et al., 2011) an approach is presented to predict the remaining flow time using an annotated transition system. The method uses two inputs: (1) transition system, and (2) an event log having events annotated with timestamps and two event transition types, i.e. *start*, and *complete*. It is assumed that the event log fits the transition system, i.e., all traces in the event log can be replayed by the transition system from the beginning to the end. During a replay, each state in the transition system is annotated with a set of time values: t, e, r, s, where t is the time the state is visited, e is the elapsed time since the start when visiting the state, r is the remaining flow time, and s is the sojourn time calculated as the difference between complete and start transactions of associated events. Assuming a large event log, there may be hundreds or even thousands of annotations per state. For each state x it is possible to create a multi-set $Q_x^{remaining}$ of remaining flow times based on these annotations. Similar multi-sets can be created for elapsed and sojourn times. Basing on these multi-sets, all kinds of statistics can be computed, e.g., the mean remaining flow time for a given state, standard deviation, minimum, and maximum. Finally, such an annotated transition system can also be used to predict the remaining time for a running case as the mean remaining flow time of all earlier process instances in the same state. Hence, the expected time of process instance completion can be derived. In the refined scenario, also standard deviation of the historic samples in the multi-set can be used to estimate the reliability of the prediction, e.g., with 90% confidence the remaining flow time is predicted to be between 40 and 45 days (Aalst, et al., 2011). The approach based on an annotated transition system is not restricted to predicting the remaining flow time (Aalst, 2011). For example, suppose that somebody is interested whether the activity g or h will occur. To make such prediction, annotation of states should include information about known outcomes for completed process instances in *post-mortem* data. For example, annotation of state x as $Q_x^{g|h} = \{0, 1, 1, 1 \ldots\}$ is created during replay by adding "0" for each visit of a process instance that includes activity g or "1" activity h. Average value of $Q_x^{g|h}$ is a predictor of the probability that a case visiting state x will be followed by activity g. In this approach based on process mining, the prediction is based on the state of the running process instance rather than some static attribute. Classical data mining approaches (e.g., based on regression or decision trees) typically use static attributes of a case rather than state information.

The approaches to *recommendations* based on process mining are discussed in detail in Section 4.2.

As three groups of methods, i.e. detection, prediction and recommendation, are envisioned for operational support, it must be noted that some methods from off-line analysis can be simply applied to analysis of partial traces, e.g., process discovery can also be applied to running process instances. In (Aalst, et al., 2010), the authors present a framework for operational support using process mining (Fig. 4.2) and detail a coherent set of approaches that focus on time information.

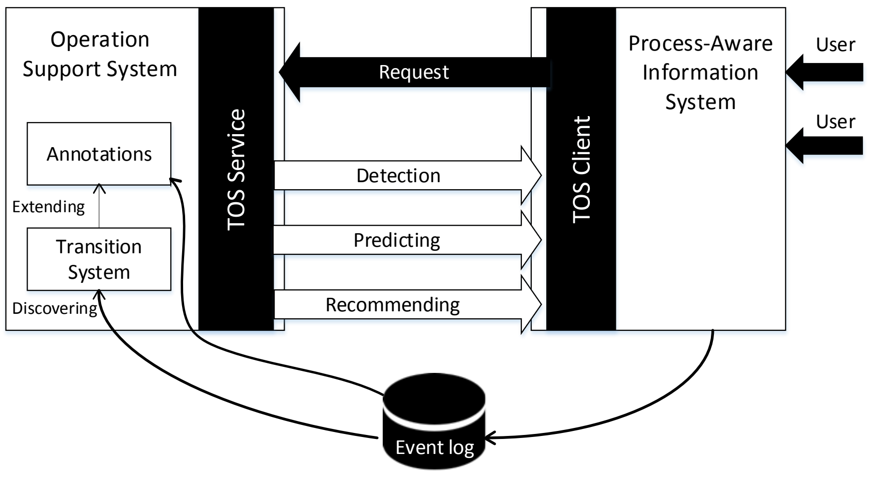


Fig. 4.2. Architecture of a system providing operational support based on time information in event logs

The proposed Time-based Operational Support (TOS) Service (Fig. 4.2) uses the transition system and other information, e.g. time annotations, to generate information about active process instances. Such information concerning active process instances is generated on request of a TOS Client. The TOS Client is rather not standalone software but an integrated part of a PAIS. Together with a request, the TOS Client sends the partial trace and currently enabled tasks of the running process instances. In response, the TOS Service generates various types of information about the current process instance. In implementation described in (Aalst, et al., 2010), the TOS Service performs at least one of three actions: (1) checks if the elapsed time of the current case is within certain boundaries of elapsed time that past process instances had in the same state, (2) predicts the remaining execution time based on the past process instances, (3) recommends the enabled activities that, in the past, led to minimum execution times. The presented approach to provision of an on-line support can be generalized. It need not to be limited to time-based support.

4.2. Process Recommendations

The general approach to recommendations based on process mining is presented in Fig. 4.3 (Aalst, 2011). The approach outlines the problem of recommendation as an extension of the prediction problem. In case of recommendations, in addition to the partial trace of the running case used in prediction methods, the set of items, i.e., enabled activities that constitute a decision space is also used.

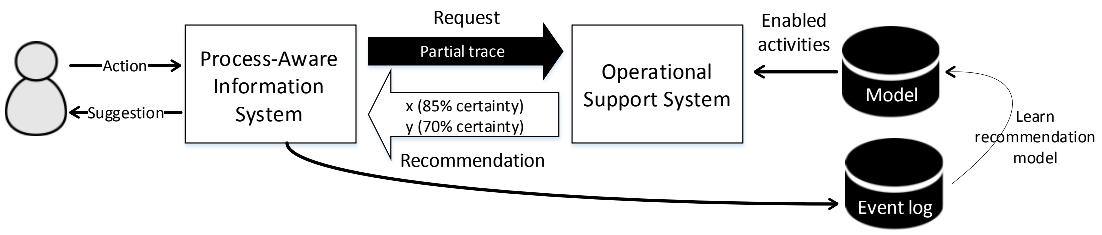


Fig. 4.3. General approach to recommendations based on process mining

The set of enabled activities is created on the basis of a process model that is either hand-made or discovered. For instance, for a partial trace presented in Fig. 4.4 and for activity *Check* that was executed as the last from the partial trace, the set of enabled activities is: $\{Inspect, Advertise, Process\}$.

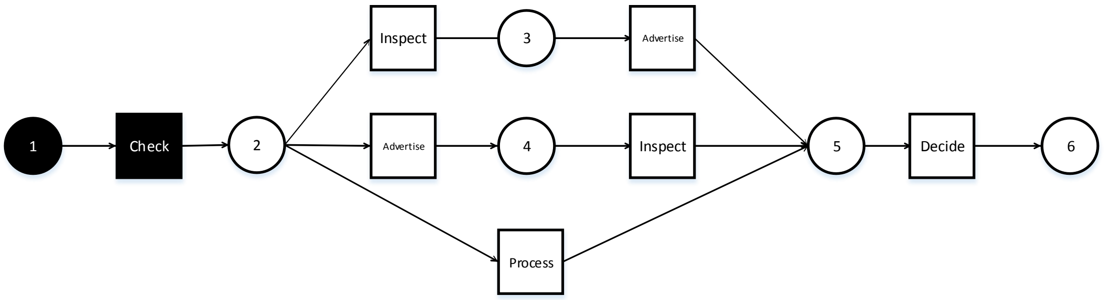
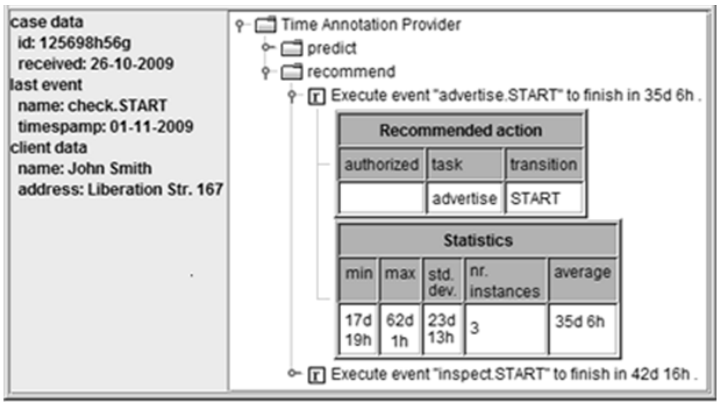


Fig. 4.4. Discovered process model represented as a transition system and a set of enabled activities for activity *e*

Best matching activity from decision space is chosen according to used rating function. In the presented approach, typically recommendation aims at minimizing the remaining flow time, total process instance cost, etc. Thus, creating a recommendation is closely related to predicting the corresponding performance indicator. Having an already executed partial trace τ, one may identify the set of enabled activities $\{a_1, a_2, \dots, a_k\}$. The partial trace is extended by each of possible enabled activities leading to creation of a number of k partial traces $\tau_i = \tau \cup a_i$ for $i \in \langle 1, k \rangle$. Each of k partial traces is then evaluated using prediction approach described in Section 4.1, i.e. two steps are performed: (1) a new trace is created by extending the partial trace of the running process instance, (2) the new trace τ_i is replayed in the transition system to identify the state to be assigned to activity a_i. Then partial traces are ranked and selected according to predicted values. In a similar way, the decision space and the recommendation problem may concern resources instead of activities. The problem then is to recommend the resource for the next activity that optimizes the value of a rating function.

A particular implementation of the approach is presented in (Aalst, et al., 2010), where time annotations created during transition system replay are used for creating recommendations concerning activities that optimize the completion time of the process instances. The ranking function based on calculation of average remaining times is used. The method is implemented as a ProM plugin. The user interface of implemented plugin is presented in Fig. 4.5 (Aalst, et al., 2010). Together with the name of a particular activity to be executed as the next one, additional statistics are provided justifying the recommendation given to the user.


case data
id: 125698h56g
received: 26-10-2009
last event
name: check.START
timespamp: 01-11-2009
client data
name: John Smith
address: Liberation Str. 167

Time Annotation Provider
predict
recommend
Execute event "advertise.START" to finish in 35d 6h .

Recommended action

authorized	task	transition
	advertise	START

Statistics

min	max	std. dev.	nr. instances	average
17d 19h	62d 1h	23d 13h	3	35d 6h

Execute event "inspect.START" to finish in 42d 16h .

Fig. 4.5. Time-based recommendations in ProM

In (Schonenberg, et al., 2008), a similar approach is presented. Recommendations for an enabled activity provide predictive information about the user goal, based on observations from the past, i.e., fully completed traces accompanied by their target value (e.g., cost, cycle time, or profit) that have been stored in an event log. The log-based recommendation service requires the presence of an event log that contains such information about process instances that have been executed for a certain process. A recommendation is initiated by a recommendation request, which consists of a partial trace and a set of enabled activities. In the presented approach for each enabled activity, two expected target values are defined:

- The expected target value when *executing* a particular enabled activity *e*;
- The expected target value when *not executing* a particular enabled activity.

During recommendation, the enabled activities are ordered according to the difference between two values, i.e., the bigger the difference, the more attractive the activity is.

In (Nakatumba, et al., 2012), the meta-model for operational support is presented. The purpose of the meta-model is to refine a common understanding of recommendation problem based on process mining and to create the common framework for development of future concepts. As in previous approaches, also this approach uses the information contained in a partial trace and a model of the running process instance to provide support to the user. The concept of recommendation, referred in the paper as *recommendation query*, is explained with the use of three simpler queries: *simple*, *compare* and *predict*. A *simple query* checks current performance of the current partial execution trace, for example, the total execution time. A *compare query* compares performance of the current partial trace to other similar traces, e.g., is the execution time of the current trace to this point higher or lower than the average. A *predict query* considers the future of traces similar to the current one. The predict query uses information concerning similar traces to provide predictions about the current trace, e.g., what is the expected total execution time for this trace. Finally, a *recommend query* gives the best possible next activity to be done based on the current partial trace. More precisely, in the proposed model the recommendation is formulated by finding all the traces being continuations of the running partial trace and finding the best matching trace that maximizes the prediction value. Formal description of the definition can be found in (Nakatumba, et al., 2012). In (Nakatumba, et al., 2012), the authors present the implementation of the operational support functionality in ProM application with desire to be used by the DECLARE system. The high level architecture of the proposed approach is presented in Fig. 4.6. A *Client* communicates with a *Workflow system*, and an *Operational Support Service* (OS Service, OSS). The Client sends one of four queries to OSS, which forwards it to a number of *Operational Support Providers* (OS Providers), which may implement different algorithms. Responses are sent back to the OSS and forwarded to the Client.

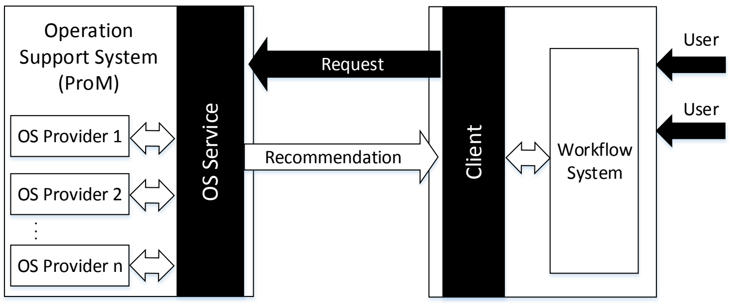


Fig. 4.6. Architecture of the operational support in ProM

The approach presented in (Schonenberg, et al., 2008) is extended in (Haisjackl & Weber, 2011) with the study of various recommendation strategies. In this approach it is assumed that the process is structured, but there is no underlying process model guiding the execution of the process instances that facilitates identification of a set of enabled activities. Instead, the set of enabled activities must be identified and is strictly connected to the problem of finding similar traces to the trace of the running process instance. Thus, in (Haisjackl & Weber, 2011) the problem of recommendation is divided in two subproblems:

1. Finding similar log traces,
2. Creating recommendation using selected strategy.

Algorithms for finding similar log traces iterates over the log to calculate the result bag for a given partial trace. Depending on the approach to identification of similar traces, a resulting bag is different (Haisjackl & Weber, 2011):

- *Prefix miner* – considers the exact ordering of activities when comparing the partial trace with a log trace; if the partial trace is a prefix of the log trace, the log trace obtains a weight of one; the result trace is coming from the trace having the highest weight;
- *Set miner* – does not consider the ordering of activities in the log, but only the presence/absence of activities; the weight is calculated by dividing the number of distinct matching activities by the number of distinct activities in the partial trace;
- *Multiset miner* – does not consider the ordering of activities in the log; it takes the number of occurrences of an activity in a log trace into account; the weight is calculated by dividing the number of matching activities by the number of activities in the partial trace; moreover, all activities from the log trace minus the activities from the partial trace are added to the result set;
- *Partial trace miner* – takes the ordering of activities into consideration but instead of comparing the entire partial trace with the log traces, it only considers the last *n* activities of the partial trace (denoted horizon); all the activities succeeding the found search trace(s) are considered result traces;
- *Chunk miner* – does not compare the entire partial trace with the log traces; instead, the partial trace is divided into chunks of size *n* (i.e., sliding window of size *n*), each of which is then compared with the log trace.

The miners are responsible for weighting log traces according to their fit with the partial trace and provide a result bag containing the mining results which is then taken by the strategies as input for generating recommendations. In particular, based on this information the strategies evaluate all the enabled activities in respect to the performance goal (e.g., minimize cycle time, minimizing error rates or maximizing customer satisfaction). The following strategies are distinguished (Haisjackl & Weber, 2011):

- *Randomized Strategy* – randomly picks one of the possible next tasks and recommends this task for execution;
- *Prefix Strategy*, *Partial Trace Strategy*, *Chunk Strategy* – strategies differ in terms of the used miner but they use the same method for calculating expected target value when *following* and *not following* a particular recommended activity; all the strategies consider the first task of each result trace from the result bag for calculating the expected values in the way described in (Schonenberg, et al., 2008);
- *Set Strategy*, *Multiset Strategy* – strategies differ in terms of the used miner; they consider all the activities of each result set in the result bag for calculating expected

target value when *following* and *not following* a particular recommended activity: the result sets which do not contain any of the enabled activities are discarded; for each possible next activity the expected target value when *executing* the activity is calculated as the weighted average of target values of all result sets containing that particular activity. The expected target value when *not executing* the activity, in turn, is the weighted average of target values of all the result sets which do not contain that particular activity.

In (Haisjackl & Weber, 2011), the miners and strategies are evaluated in terms of performance. No strategy is always outperforming all other strategies. Consistently good performance is delivered by the Prefix Strategy. In the majority of cases, both the Partial Trace Strategy and the Chunk Strategy are outperformed by the Prefix Strategy. However, the Chunk Strategy might bear some potential for processes comprising loops.

In (Dorn, et al., 2010), an approach to context-sensitive, self-adjusting process recommendations is presented. The paper focuses on two major challenges associated with processes that share characteristics with VO collaborative processes (Dorn, et al., 2010):

- Users in people-driven processes require a combination of personalized recommendations, while exploiting the best practices emerging from the overall user community;
- Flexible processes need to evolve across time to reflect the changes in working style, business constraints, and impact of cross-organizational cooperation.

The recommendation is performed for processes which have models guiding execution of process instances (Fig. 4.7) (Dorn, et al., 2010). Despite existence of a model, a user is able to select activities in any order. The process model describes a generic process that provides a rough guide for most cases. The proposed recommendation system monitors user's decisions and continuously adapts to recommend always the most suitable next activities. The approach relays on identification of sequence graphs. The sequence graph *SG(P,E)* consists of nodes representing the individual process activities *P* and a set of edges *E*. A directed edge $e \in E$ in *SG* between two nodes *A* and *B* describes a temporal sequence: activity *B* follows immediately after *A*. Whenever a user performs activity *B* after *A*, the edge value is increased. The *SG* accumulates all individual activity sequences for a particular process. Thus, it yields the preference of following a particular path through the process. Identification of sequence graphs is performed during process instance execution on the basis of event log analysis. Sequence graphs are constantly extended as subsequent process instances are executed.

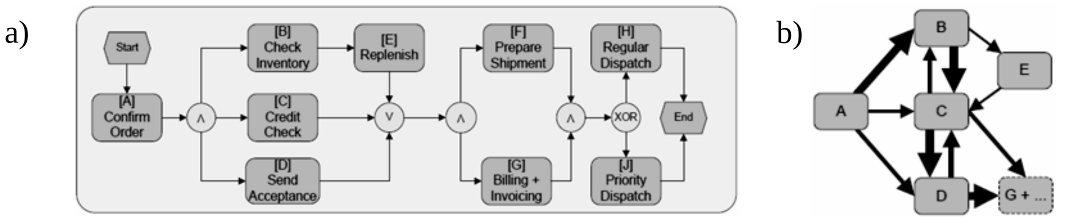


Fig. 4.7. The process model (a) and the corresponding sequence graph (b)

Created sequence graphs are used during process instance executions by the Process Instance Manager component. The Process Instance Manager is responsible for:

- Selecting relevant activities to be performed in the current process instance, where the selection is based on information about current process instance state and information from the sequence graph;
- Tracking user activities that have been completed;
- Providing a list of activities that are ready to be carried out;
- Analyzing process instances for skipped and out-of-order activities.

Personalized recommendations based on individual sequence graphs yield highly relevant activity rankings. Drawbacks of such personalized recommendations are the following:

- Recommendations are limited to activities that a particular user has executed so far;
- Alternative sets of activities that potentially reduce overall processing time remain unavailable;
- Recommendations are not applicable in exceptional situations that have not been encountered by the user before;
- Recommendations reinforce inefficient or even incorrect activities that the user tends to execute.

Crowd-based recommendations mitigate this shortcoming. Crowd-based recommendations enrich the set of relevant possible process paths through aggregation of the process experiences from multiple users. The mechanism of creating recommendations is presented in Fig. 4.8 (Dorn, et al., 2010):

1. Incoming request for recommendation triggers the recommendation mechanism,
2. The recommendation system collects information from the Process Instance Manager;
3. The recommendation system collects information from both personal and crowd-based sequence graphs;
4. The recommendation system subsequently provides the user with the recommended activities;
5. The user selects one activity and performs it; the Recommendation System Monitor observes the user’s actions (5a), and other events (5b) to determine the true process progress;
6. The Recommendation System Monitor updates the Process Instance Manager whenever an activity is completed;
7. The Process Instance Manager updates the personal and crowd-based sequence graphs for each completed step;
8. In regular intervals, the Process Miner takes a sequence graph and generates an updated process model.

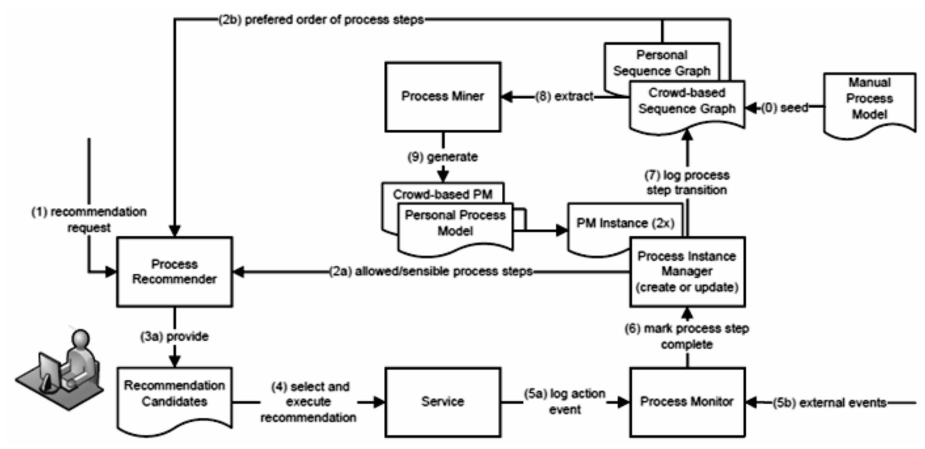

(2b) prefered order of process steps
Personal Sequence Graph
Crowd-based Sequence Graph
Manual Process Model
Process Miner
(8) extract
(0) seed
(9) generate
(7) log process step transition
Crowd-based PM
Personal Process Model
PM Instance (2x)
(1) recommendation request
Process Recommender
(2a) allowed/sensible process steps
Process Instance Manager (create or update)
(3a) provide
(6) mark process step complete
Recommendation Candidates
(4) select and execute recommendation
Service
(5a) log action event
Process Monitor
(5b) external events

Fig. 4.8. Feedback cycle for personal and crowd-centric recommendations

The overall recommendation combines user-centric and crowd-based recommendations according to the classifier α. The classifier describes the user on a scale between 0 and 1, where 1 denotes a user always adhering to his individual work style – the eagle. At the other extreme end of the classifier ($\alpha = 0$), a user follows generally applied work practices – the flock. The classifier α is determined for each user and process as a user's work style potentially deviates for one process to another. The overall recommendation merges user-centric and crowd-based recommendations according to the following formula: $\mathrm{R}_{\mathrm{overall}} = \alpha * \mathrm{R}_{\mathrm{user}} + (1 - \alpha) * \mathrm{R}_{\mathrm{crowd}}$. The value of α increases when the user carries out activities that originated from R_{user}. Similarly, the value of α is reduced when the user follows crowd-based recommendations. The merged recommendations coming from individual and crowd sourced are sorted according to overall recommendation value. A dynamic adjustment if α value reflects a user's learning effects and his/her adaptation to changing process. The move of α value is determined by process output, i.e., process success. When α remains close to 1, but process success declines, it is assumed that the personalized recommendations fail as they most likely reinforce bad decisions. In this case, α is pushed towards the neutral value to take again into account crowd-based recommendations.

In (Swinkels, 2012), intelligence of organizations, instead of single users is used to better formulate recommendation. The architecture of the suggested system is presented in Fig. 4.9. The event logs of various organizations are integrated into one so called Collective Event Log (CEL). The Collaborative Event Log is used to mine Collective Transition System (CTS) describing behavior of organizations captured in the CEL. A Multiple Log Recommender (MLR) is a central component of the suggested system. When recommendation request is set by an organization, MLR uses information provided in the request and information coming from CTS to provide recommendations. CTS provides to MLR a mined time-annotated transition system. MLR creates further continuation of process instances. The best continuation of a process instance is returned to the client.

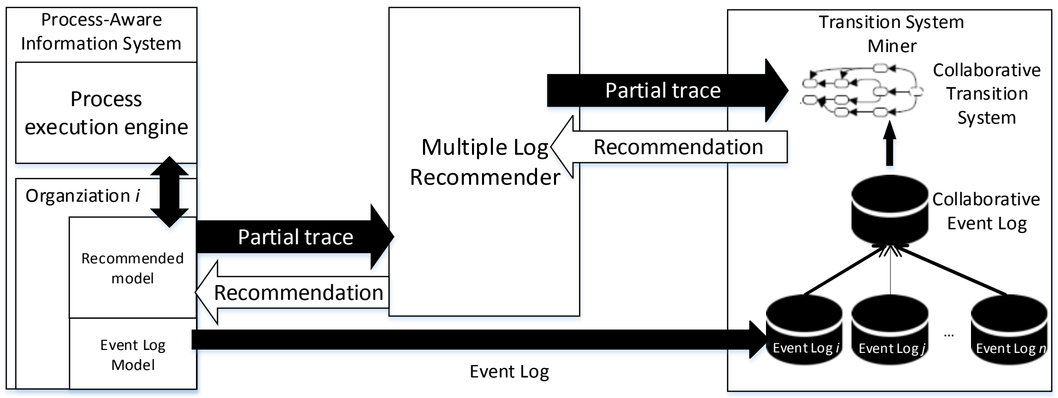


Fig. 4.9. High-level architecture of the collective recommendation system

A request for a recommendation by a process instance of an organization contains: (1) the process instance identifier, which is a unique number for each process instance, (2) a list of activities that were executed since the last recommendation request, and (3) a list of possible activities that can be executed next. Extended number of parameters passed to the recommendation system is greater than number proposed in (Aalst, et al., 2010) and in (Schonenberg, et al., 2008), because the recommendation service has to recommend an activity which is present in the list of all the possible activities allowed in organizations requesting recommendations. A recommendation cannot involve an activity that is not available or possible in the organization.

Another approach is proposed by in (Almeida, et al., 2004) where recommendations are based on ontology and semantic rules that generate possible process alternatives or suitable activities if execution of a workflow instance fails to proceed. Among other works, the prediction engine of Staffware (Staffware, 2003) uses simulation to complete audit trails with expected information about future activities. The approach does not provide a means of learning to make better predictions over time. A more refined approach focusing on transient behavior is presented in (Rozinat, et al., 2009). In (Stoitsev, et al., 2007) recommendations are based on best-practices shared by users within a company. This approach supports operational decision making using process mining techniques and simulation in the context of YAWL. In (Weber, et al., 2004) the case based reasoning approach is presented. The prototype implementation of CBRFlow is able to adapt a process model to changing situations at run-time and provide the workflow system with learning capabilities. Recommendations can also be based on a Product Data Model as discussed in (Vanderfeesten, et al., 2008), but they are specific for product based workflows. In (Barba, et al., 2012), the authors propose a recommendation system based on a constraint-based approach extended to consider not only the control-flow, but also the resource perspective in order to optimize performance goals of business processes.

4.4. Mining Behavioral Patterns in Collaboration

Identification of behavioral patterns in unstructured processes (*cf.* Section 3.1.2) is achieved via process mining methods, classic data mining methods, and social network analysis.

Process mining methods can be split into five categories: (1) generalization and clustering, (2) constraint-based analysis, (3) variant analysis, (4) social network mining based on event data, and (5) context capturing.

Majority of process mining methods were developed for discovery and analysis of structured process. Applying these methods to unstructured processes is followed by high complexity of discovered models, and low understandability and usefulness of such models. Thus, the main objective of process mining methods developed for analysis of unstructured processes is to lower complexity of discovered models. In *fuzzy mining* methods the concept of generalization of activities and transitions among activities is proposed. Generalizations combine activities frequently appearing together (Gunther & Aalst, 2007). The method proposed in (Bose & Aalst, 2009) is based on initial clustering of process instances according to their similarity. Then classic process model discovery algorithms are applied separately for each cluster. In (Bose & Aalst, 2009) a generic edit distance metric is proposed that captures dependencies among activities and takes into account various degrees of similarity among activities. Generic edit distance metric eliminates some drawbacks of *Hamming* and *Levenshtein* distance metrics. Classic similarity metrics are used (Delias, et al., 2013) to count similarity among traces of process instances, e.g., *cosine* similarity is used for clustering of healthcare processes. Some approaches to analysis of behavior of service consumers and service providers are presented in (Aalst, 2004) (Aalst, 2006) (Aalst, 2013).

Traditionally mined descriptive process models aim at specification of behavior that is allowed in a process. As mentioned above, in case of unstructured processes this approach leads to very sophisticated models. On the contrary, constraint-based approaches specify only what is forbidden during process execution. This group of methods is implemented as constrain-based workflows and constraint-based languages such as DECLARE (Aalst, et al., 2009). Process mining methods discover constraints that are used to guide the execution of process instances. Other approaches to unstructured processes rely on analysis of variants of execution of process instances. Variants describe common ways of process execution (Rozinat, 2013). Variants do not aim at achievement of full process models.

While majority of methods concentrate on a process perspective, i.e., a control flow perspective, some methods discover social perspective by analysis of information concerning resources and structure of activities. As a result, social relations describing handover of work (Aalst, et al., 2005) or organizational structure (Song & Aalst, 2008) are discovered. While usually social and control flow perspective are analyzed separately, new approaches to analysis of human behavior using process mining claim the need of simultaneous analysis of more than one perspective at time (Leoni & Aalst, 2013). However, such methods miss important aspect of analysis of correlation among various perspectives. For instance, analysis of influence of structure of discovered social network on the model is not possible. All the methods aiming at discovery of a social perspective assume a limited scope of available data that are associated with events in an event log. Thus, a set of social relations to be discovered is very limited. On the contrary, in many modern PAIS, each event is usually described by a number of attributes referring both to activity itself as well as resources involved in activity execution and can be additionally supplemented with data from other systems. Up to date, no approach to identification of social roles and social relations based on such rich event logs has been proposed.

An approach to capturing a context of human behavior is presented in (Ghattas, et al., 2009). Context is represented by a set of attributes describing a process instance. A presented case study refers to clinical processes. An example of a process instance context includes the following attributes: patient age, gender, race, chronic illnesses, general mental and overall state of the patient, etc. Such process instance context is used to build a decision tree. So far, the notion of context is limited to process instance context. Up to date, the notion of activity context, which is crucial for providing recommendations for the running process instances, has not been defined and used.

Analysis of human behavior goes far beyond process mining techniques. In classic data mining a wide set of sequence mining techniques has been developed over years. Among others, sequence mining algorithms take into account activity ordering. This group of algorithms include *AprioriAll* (Agrawal & Srikant, 1995), *GSP* (Srikant & Agrawal, 1996), *SPAM* (Mane, 2013), and *BIDE* (Wang & Han, 2004). For instance, *PrefixSpan* algorithm (Pei, et al., 2004) is based on a pattern-growth approach to discovery of sequential patterns. Its optimized versions include bi-level projections of sequence database and pseudo-projection. In general, data mining algorithms have limitations preventing them from being used in VO collaborative process model discovery: (a) they focus on frequent behavior without trying to generate models, (b) they cannot model choices, loops, etc., (c) they cannot well handle concurrency, (d) they do not include analysis of patterns on the level of activity attributes, i.e., only activity names are considered, (e) they consider local patterns only, i.e., no overall process model is created. Some methods deal with individual limitations, e.g., discovery of parallel patterns was introduced in (Hwang, et al., 2004). Three algorithms were proposed do discover so called *temporal patterns* capturing *followed* and *overlapped* relations among activities: *TP-Graph*, *TP-Itemset*, and *TP-Sequence*.

Classic data mining techniques such as *classification* and *clustering* (Witten, et al., 2011) (Wiszniewski, 2011) are used by many process mining methods. For example, in (Delias, et al., 2013) first clustering and then classification are used to reason about context-aware patterns in patient care. Decision trees are used to explain reasons underlying particular decision making points in a process (Rozinat & Aalst, 2006).

Social network analysis (Watts, 2004) concentrates on analysis of human behavior from the perspective of social network structure and its dynamism. Recent works in the area of social networks include works on evolution of social networks (Jeong, et al., 2003) (Morzy, 2013), resilience of networks (Brendel & Krawczyk, 2008) (Brendel & Krawczyk, 2010), propagation of information (Pastor-Satorras & Vespignani, 2001), categorization of social networks (Morzy, 2012), evaluation of actors reputation and importance (Morzy, et al., 2009) or their negative impact on the network (Krawczyk & Brendel, 2006), discovery of social aspects in incomplete or aggregated data sets (Morzy & Forenc, 2013). Social network analysis abstracts from activities and processes performed by members of a social network. Thus, existing methods for social network analysis do not support the analysis of correlation between processes performed by collaborators and a structure of underlying social network.

4.6. Event Log Formalization

In this section, the concepts of event, trace and event log are formally defined.

Definition 4.1. (Event). An *event e* is an object describing an observation at time *t*, where this observation is related to precisely one activity instance executed within a collaborative process instance.

Let E^* denote the set of all the events.

Each activity instance executed within a process instance is related to at least one event. Within a VO collaborative process instance, events may refer to the same activity instance (e.g., activity start or completion) or to different activity instances of the same activity (e.g., activity executed in a loop).

For each event $e \in E^*$ and attribute $a = \langle an, av_{an} \rangle$ with the attribute name $av_{an} \in AV$, $e(an)$ is a value *av* of attribute a for event *e*.

Definition 4.2. (Trace). A *trace* $\tau_p = \{e_0, e_1, \dots, e_{i-1}\}$ is a finite sequence of events recorded for a VO collaborative process instance *p*, where $|\tau_p| = u$ and $\tau_p(u) = e_u$.

Example 4.1. An example of a trace is presented in Tab. 4.1. Each row corresponds to an event, where each event is described by four attributes presented in columns: *activity name*, *resource*, *timestamp* and *cost*. ▪

Tab. 4.1. Trace $\boldsymbol{\tau(p)}$

Event	Activity name	Resource	Timestamp	Cost
e_0	A	Tom	01.12.2012 3.15 p.m.	30
e_1	A	Tom	01.12.2012 3.19 p.m.	40
e_2	C	Kate	01.12.2012 3.21 p.m.	10
e_3	B	Paul	01.12.2012 3.35 p.m.	10
e_4	B	Paul	01.12.2012 4.02 p.m.	40
e_5	C	Kate	01.12.2012 4.15 p.m.	20
e_6	D	Tom	01.12.2012 4.16 p.m.	20

Let $E = \{e_0, e_1, \dots, e_q\}$ denote a sequence of *q*+1 events.

Let $E_p(vi_i)$ denote a set of events assigned to activity instance vi_i from VO collaborative process instance *p*. Then, $E_p(vi_i) \subseteq \tau_p$.

Let $A(\tau_p)$ denote a set of process instance attributes.

Definition 4.3. (Partial trace). A *partial trace* $\tau(p)$ is a trace corresponding to a collaborative process instance that is still running.

Example 4.2. In Fig. 4.10, a partial trace of a collaborative process instance *p* is presented. The dark activity *E* indicates that instances of activities *A*, *B*, *C* and *D* have already been executed. Execution of those activity instances was recorded in trace $\tau(p)$. Activities *E* and *F* have not been reached yet which means that process instance *p* is still in progress and trace $\tau(p)$ is partial. Each event *e* from trace $\tau(p)$ is associated with exactly one executed activity instance. Each activity instance is assigned to at least one event. For example, activity A is associated with event e_1 and e_2, i.e., $E_p(A) = \{e_1, e_2\}$. ▪

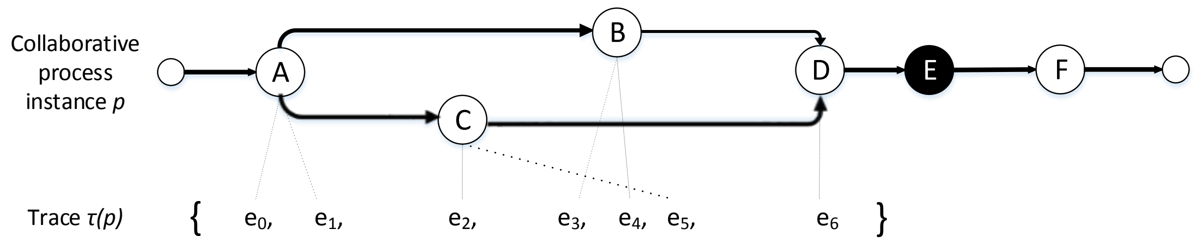

Collaborative process instance *p*
A
B
C
D
E
F
Trace *τ(p)*
{ e0, e1, e2, e3, e4, e5, e6 }

Fig. 4.10. Collaborative process instance and its partial trace

Let $\tau(p, e)$ denote a subsequence of events from trace $\tau(p)$ that were recorded before event e inclusive.

Example 4.3. For trace from Example 4.2, $\tau(p, e_3) = \{e_0, e_1, e_2\}$. ▪

Definition 4.4. (Event log). An *event log* ℓ is a set of traces.

Let $E(e, t_1, t_2)$ denote a set of events different than e that were recorded in an event log in time period from t_1 to t_2 inclusive.

5. Conceptual model of the RMV method

5.1. RMV Method Requirements

Basing on unpredictable and emergent character of VO collaborative processes (*cf.* Section 3.1.2), nine main requirements for computer support for VO collaborative processes are defined in this dissertation:

1. *Guidance for process instance execution* – computer support for VO collaborative processes should provide whenever possible a guidance for VO collaborative processes instance execution; guidance should be performed by indication of the next activities and collaborators involved in their execution; in case when the set of best fitting collaborators is impossible to determine, it should provide information vital for their selection (*cf.* Section 3.1.2);
2. *Support for conformance analysis* – once guidance is given, computer support for VO collaborative processes should provide means for verification of conformity of process instance execution to the guidance: potential discrepancies should trigger adaptation actions and influence a confidence of future recommendations;
3. *Support for adaptation and flexibility* – computer support for VO collaborative processes should allow collaborators to respond to new situations; computer support should enable selection and modification of sets of collaborators and performed activities (*cf.* Section 3.1.2);
4. *Descriptive model* – as VO collaborative processes are frequently modeled ad-hoc (*cf.* Section 3.1.1), computer support for VO collaborative processes should rely on descriptive models of collaboration, instead of prescriptive ones; descriptive models represent actual, real behavior of collaborators (*cf.* Section 4.1);
5. *Computer supported approach* – a computer supported approach should provide relevant information for the collaborators, instead of making decisions on behalf of them; the decision making should remain under responsibility of collaborators; VO collaborative process instance execution is influenced by rules of social norms and shared tacit knowledge on collaboration, which are elements that may hardly be taken into account by information systems (*cf.* Section 3.1.2);
6. *Collaborative wisdom* –guidance for VO collaborative process execution should be based on the best practices following from the overall collaborators community; to exploit the wisdom of community, the reasoning performed to provide operation support should rely on analysis of behavior of all the collaborators involved in various VO collaborative process instances (*cf.* Section 4.2);

7. *Reusability* – computer support for VO collaborative processes should provide reusable outcomes to rule collaboration within various VOs;
8. *Social aspect and context* – efficient computer support of VO collaborative processes must include modeling and analysis of social relations among collaborators (*cf.* Section 3.1.3) and context in which a VO collaborative process instance is executed (*cf.* Section 3.2);
9. *Continuous instantiation* – computer support for VO collaborative processes should include selection of actors based on aspects relevant for VO collaborative processes; as the set of activities and collaborators is dynamic, the instantiation should be performed throughout the execution of VO collaborative process instance (*cf.* Section 2.3).

5.2. RMV Method Motivation

The needs of computer support for VO collaborative processes go beyond possibilities of existing methods. The methods in the area of PAIS, process mining, recommender systems and VO creation support do not satisfy the above requirements.

Various PAISs aim at finding right balance between the support offered to a user on the one hand, and flexibility in definition and execution of user's activities, on the other. The trade-off between support and flexibility is presented in Fig. 5.1 (Dumas, et al., 2005). In particular, three representative types of PAISs are distinguished: production workflow systems, ad-hoc workflow systems, and computer supported collaborative work systems.

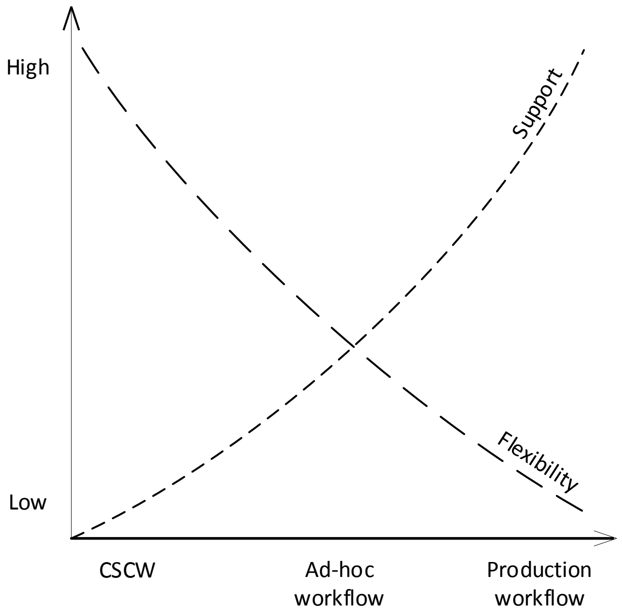


Fig. 5.1. Trade-off between flexibility of process definition and support for human actions in PAIS

In the *production workflow managements* systems, process models highly restrict the collaboration limiting flexibility. The concept of a process model, which is at the core of workflow management systems, aims at defining the complete set of potential interactions within a given process as precisely as possible, i.e. design and run-time phases of process lifecycle are precisely separated.

A possibility of process model modification at run-time is provided by *ad-hoc workflow systems*. However, ad-hoc workflow systems require a system user, i.e., a collaborator, to declare at some stage the future shape of his/her process. This declaration may concern an initial process model to be later modified (adaptive processes) or some process parts to be later used during process reconfiguration (flexible processes).

The *computer supported collaborative work (CSCW)* systems do not strongly restrict collaboration with predefined sets of activities. Instead, activities are rather considered as functions that are available for the collaborators, with weak constraints on the precedence of some activities. The support focuses on communication among collaborators giving poor guidance for the process instance execution.

Therefore,

> *there is a need for a method that maximizes the scope of guidance for collaborative process execution (requirement 2, 3, 5) with a little negative impact on flexibility of collaborative process instance execution (requirement 1), where a set of available activities and a set of involved collaborators are unpredictable and dynamic (requirement 4).*

The problem of discovery of descriptive process models is addressed by process mining methods. In particular, process mining offers methods for operational support that provide guidance during process instance execution. However, process mining methods miss significant aspects preventing them from being used in support of VO collaborative process executions:

- *Support for structured processes* – discovery of process models and activity sequences is limited to structured processes, while VO collaborative processes are quasi-structured (*cf.* Section 4.2);
- *Required knowledge of process model* – recommendations are provided on the basis of some underlying workflow model (discovered or hand-made); in case of VO collaborative processes the full model of a process cannot be discovered or modeled (*cf.* Section 4.2);
- *Limited context* – in majority of methods, recommendations are based on information concerning the activity itself; as evolving character of VO collaborative processes follows from constant analysis of process or activity context, efficient operational support for VO collaborative processes must take context into account (*cf.* Section 3.2); some approaches to inclusion of context take into account process instance properties which are used for recommendation of full process instance traces, but not for recommendation of particular activities in a particular process instance state (*cf.* Section 4.3);
- *Focus on activity names* – methods mining behavioral patterns in unstructured processes permit identification of activity names, but no other aspects of a process which are significant from the efficient collaboration point of view, e.g. social aspects; thus any guidance concerning process instance execution is formulated on an abstract level rather than executable one (*cf.* Section 4.3);
- *Low-level recommendation criteria* – criteria functions used when providing guidance during process instance execution are defined on a low level, i.e., execution time or cost; it is impossible to define goals on a business level, because the analysis of correlation between process instance structure and successful or unsuccessful process completion is not performed (*cf.* Section 4.2);

- *Inflexible preferences* – operational support misses the aspect of collaboration and flexible definition of collaborator preferences (*cf.* Section 4.2).

Recommender systems other than ones based on process mining techniques, provide some guidance concerning activities performed by recommender system users. However, support for the concept of context is rather limited, i.e., the context is static and does not change over time.

Therefore,

> *there is a need for a method that identifies descriptive models of a VO collaborative processes (requirement 4) and their context (requirement 8) in multiple past executions of VO collaborative process instances (requirement 6), and provides context-aware, business-oriented operational support for future VO collaborative process instances (requirement 7).*

Existing methods of partner and service selection during VO creation, do not address the characteristics of inter-organizational collaboration (*cf.* Section 2.3). This leads to the following shortcoming of the methods preventing them from effective application to collaborative processes:

- *Required knowledge of the process model* – it is assumed that the model describing collaboration is known in advance and that it does not change;
- *One-time selection* – all the partners are selected before a VO collaborative process starts, while in practice new partners may be required when the process progresses, or some initially selected partners may appear useless, so the evolving aspect of the partner selection is not addressed,
- *Unsupported social aspects* – a combined analysis of potential partners, their services and social relations is not supported.

Therefore,

> *there is a need for a method of instantiation of VO collaborative processes such that the selection is performed throughout the VO collaborative process instance execution (requirement 9) and selection of actors is based on criteria analysis relevant for VO collaborative processes (requirement 8).*

5.3. RMV Method Outline

The main idea of the *Recommendation Method for Virtual Organizations* (RMV method) is an automatic discovery of activity patterns and ad-hoc generation of recommendations for VO collaborative processes, where a VO collaborative process is performed within a SOVOBE. An *activity pattern* is a service protocol (*cf.* Section 3.1.3) that is frequently performed in a particular contexts as a part of various instances of VO collaborative processes of the same type. Formal definition of activity pattern is provided in Section 5.4.4. Discovered activity patterns are used to provide on-request recommendations for the running VO collaborative process instances. Selection of activity patterns for recommendation is based on similarity between activity pattern contexts and the context of running VO collaborative process instance. The best matching activity pattern is instantiated before actually being incorporated into VO collaborative process execution. Selection of the best matching activity pattern from a set of recommended activity patterns and its instantiation is performed in a collaborative way by a group of collaborators referred to as *selecting collaborators*.

5.3.1. RMV Method Steps

The high level architecture of the approach proposed in this dissertation is presented in Fig. 5.18. There are two main components of the architecture: PAIS and *Operational Support System* (OSSys). PAIS functionality is used by collaborators to execute activity instances within VO collaborative process instances. Potentially, PAIS supports execution of multiple VO collaborative process instances at the same time. OSSys provides operational support for execution of VO collaborative process instances by their analysis and formulation of recommendations. OSSys is the actual implementation of the RMV method.

Two components take part in communication between OSSys and PAIS: *Operational Support Service* (OSS) and *Operational Support Client* (OSC). OSS provides access to OSSys functionality. OSC is a part of the PAIS that is responsible for communication with OSS. A request send by OSC is handled by OSS. The OSS forwards the request to OSSys functional components. Responses generated by OSSys functional components are sent back to the OSS and forwarded to the OSC.

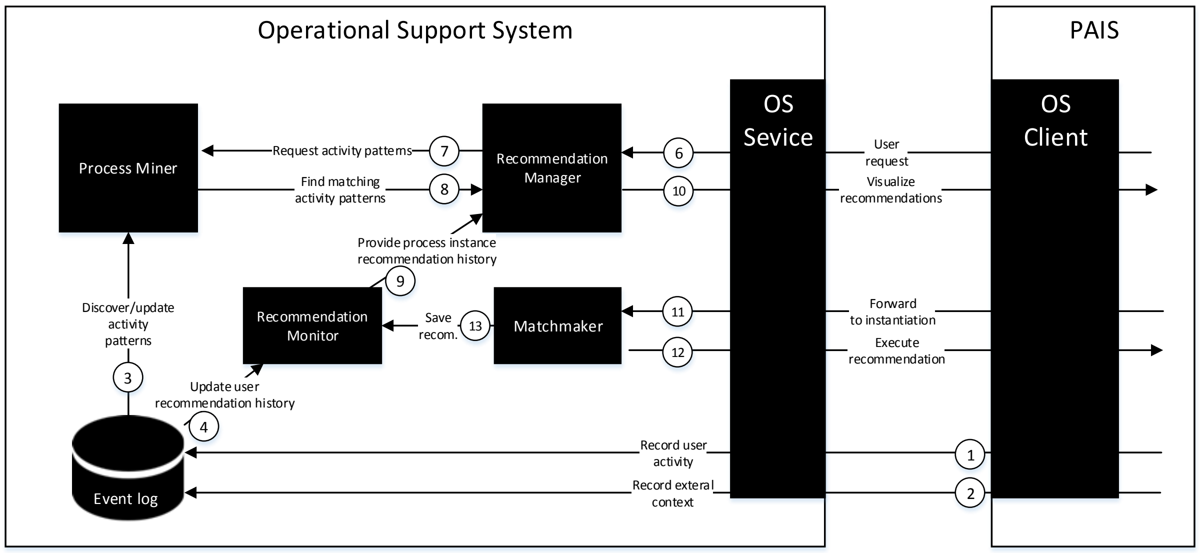


Fig. 5.2. Main steps of the RMV method

The RMV method consists of the two main phases:

1. *Discovery of activity patterns and their context* (steps 1-3 in Fig. 5.18) – in this phase the set of activity patterns is discovered; discovery requires separation of VO collaborative processes parts that appear in quasi-structured processes: (a) are repeatable and occur in many VO collaborative process instances are repeatable and occur in many VO collaborative process, (b) are unique and have ad-hoc character; the discovered repeatable parts are stored as activity patterns associated with contexts they appeared in;
2. *Recommendation formulation* (steps 4-13 in Fig. 5.18) – in this phase, activity patterns suited to a particular context of VO collaborative process are selected and recommended for inclusion in further execution of a process instance; once activity pattern is selected, it is instantiated before being incorporated into VO collaborative process execution; the second phase is performed on request. The request for

recommendation of activity patterns is generated manually by a collaborator or automatically by PAIS each time a change is observed by the OSC in the VO collaborative process instance.

In the first phase, information about activity instances that are executed by collaborators is stored in the *Event log* (step 1). Information concerns activity instances in both completed and still ongoing VO collaborative process instances. The information about activity instance in the Event log encompasses the context of the activity instance execution (2). On the basis of data stored in the Event log, *Process Miner* performs discovery of activity patterns and their context (3). The discovery is performed in preconfigured intervals. Each activity pattern discovery step updates the set of activity patterns available for operational support. Each activity pattern is stored together with information concerning contexts it was discovered in.

In the second phase, a *recommendation formulation* includes: finding the best matching activity pattern, and instantiation of the activity pattern. The VO collaborative process instance within which a recommendation of an activity pattern was made is referred to as the *selected process*. Selected and selecting processes are interrelated and are executed simultaneously. Collaborators involved in a selecting process, i.e., selecting collaborators, are in general different from actors executing the service protocol being instantiated. In the selecting process, recommendation is performed as a response to a request (6). The request includes current context of a collaborator and his/her preferences concerning the recommendation. The request is first passed to the Process Miner (7) and initial set of matching activity patterns is selected (8). Final selection of best matching recommendations is made by the *Recommendation Manager*. The Recommendation Manager performs the selection in real-time, each time a request is received. During the selection, the Recommendation Manager uses:

- Information provided by the Process Miner;
- Information provided by the Recommendation Monitor (9); the Recommendation Monitor collects information concerning history of recommendations accepted by a selecting collaborators and analyzes the usage of recommendations in VO collaborative process instances (4).

The Recommendation Manager presents one or more of the best matching recommendations to selecting collaborators (10). The selected recommendation is submitted to the *Matchmaker* for instantiation (11). Information about the newly selected and instantiated activity pattern is saved in the Recommendation Monitor (13). Finally, instantiated activity pattern is returned to the PAIS for the use and monitoring in the VO collaborative process instance. The recommendation can be used by the OOC in various ways. A simple scenario may concern information being just displayed to collaborators for their recognition. In more advanced applications, recommended activity patterns are instantiated and passed to the workflow engine that supports flexible definition of workflow processes. A sequence of recognized and instantiated activity patterns can potentially create a fully executable VO collaborative process instance. Finally, conformance rules generated on the basis of classes of service entities and service requirements are used for monitoring events generated by further execution of a process instance. The fact that further execution of a VO collaborative process instance is in line with recommendation, i.e., conformance rules are satisfied, increases the confidence indicator of a recommendation. In this way, the RMV method is a full framework for discovery, recommendation, instantiation and verification of activity patterns – Fig. 5.3.

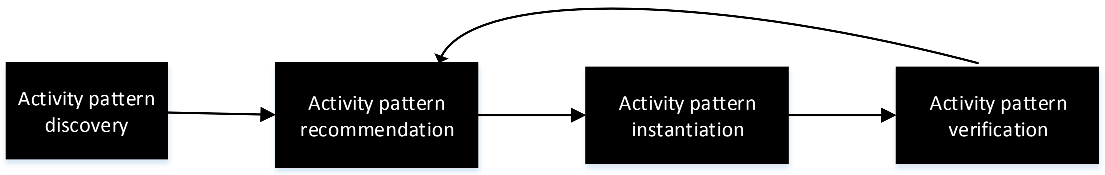

Activity pattern discovery
Activity pattern recommendation
Activity pattern instantiation
Activity pattern verification

Fig. 5.3. Activity pattern life cycle

Recommendations are based on selecting collaborators' context, thus activity patterns can be used in many VO collaborative process instances of various process types as far as they match selecting collaborators' context. This allows the knowledge to be shared between multiple and various VO collaborative process instance executions, as wall as various VOs. Tab. 5.1 summarizes functionality of the main OSSys components.

Tab. 5.1. Functionality of the RMV method components

Component name	Functionality
Process Miner	• Identification of activity patterns • Identification of activity pattern contexts • Storage of activity patterns and their contexts
Recommendation Monitor	• Analyzing collaborators' activities for skipped, out-of-order and out-of-scope activities that do not follow generated and instantiated recommendation using conformance rules • Analyzing collaborators' tendency to follow recommendations
Recommendation Manager	• Analysis of information from the Process Miner • Creating a ranked list of recommendations for selecting collaborators
Matchmaker	• Instantiation of activity pattern by selection of best matching collaborators and services
Event Log	• Recording events corresponding to collaborators' activities • Recording contextual information and VO collaborative process instance attributes

5.3.2. Key Concepts of the RMV Method

This section describes in a general way key models and techniques used in the RMV method. Formal definition of models is presented in Section 5.4. The formalization of methods is presented in Section 5.5.

Eight key aspects related to context-aware recommendation of activity patterns for VO collaborative processes in the RMV method are:

1. Activity pattern representation;
2. Scope of information required in a VO collaborative event log;
3. Scope and functionality of SOVOBE services;
4. Context model;
5. Activity pattern discovery technique;
6. Recommendation generation technique;
7. Activity pattern instantiation technique; and
8. Parameterization of the RMV method.

1. Activity pattern representation

In the RMV method, the SOA approach to VO collaborative process execution is assumed (*cf.* Section 3.1.3), i.e., execution of each activity in a VO collaborative process instance is associated with a triple: service consumer, service interface, and service provider. Each discovered activity pattern is represented by a service protocol. Service protocol is a bi-perspective model of process (*cf.* Section 4) that captures relations between two perspectives: *control flow* perspective (service protocol process model and service-oriented summary of a process model) and *social* perspective (service network schema and social network). The formal model of activity patterns is presented in Section 5.4.4. The formal model of service protocol is presented in Appendix A.

The RMV method is independent from the specific formalism used to describe the control flow of a process. In the prototype implementation presented in Section 6.5, temporal graphs are used for representation of a process model of an activity pattern. Activity pattern process model enables both sequential and parallel representation of activities ordering.

2. Collaborative process event log

Discovery of activity patterns includes identification of: (1) contexts of an activity pattern, (2) the set of service entities, (3) the set of classes of service entities, (4) the set of service requirements and (5) mapping between all those elements. Discovery of activity patterns is possible only if relevant data are available in the event log. Formalization of VO collaborative process event log is presented in Section 5.4.3. In addition to data usually available in an event log (*cf.* Section 4.5), the following information about each event is expected:

- *Service entity attributes* – a non-empty set of attributes that describe service entities involved in activity instance execution associated with the event;
- *Social attributes* – a non-empty set of attributes that describe service relations among service entities.
- *Context attributes* – a non-empty set of attributes that describe the context of event occurrence, i.e., circumstances of event generation. The scope of context information to be gathered is shared by all the events. Changes in particular values of context attributes are recorded throughout the VO collaborative process instance execution. In this way, context attributes capture the dynamic aspects of VO collaborative process instance context. No assumptions concerning the scope of information provided as the context attributes and process instance attributes are made. The scope of information useful for a particular VO collaborative process instance depends strongly on the nature of the process.
- *Service entity identifiers* – a non-empty set of identifiers of service consumer, service provider and service interface.

A rich VO collaborative processes event log is currently commonly available in many modern PAISs. Moreover, an event log can be additionally enriched by data coming from other information sources such as social media. VO collaborative process event log may represent high level activities performed by organizations, as well as lower level activities describing interaction among information systems.

3. SOVOBE’s services

SOVOBE’s services provide an access to data necessary for creation of a VO collaborative process event log.

In the RMV method, each VO collaborative process is executed within SOVOBE, i.e., collaborators use services provided by SOVOBE throughout the VO collaborative process instance execution. SOVOBE logs information concerning activities performed by SOVOBE members within various VO collaborative processes. SOVOBE’s services also provide an access to these logs and other data including:

- Context information,
- Service entity descriptions,
- Service relations descriptions,
- The set of activity instances in running and past collaborative process instances executed by various VOs,
- Assignment of service entities to activity instances, including terms under which organizations collaborate.

The RMV method itself is also one of SOVOBE’s services provided to various VOs in a form of an OSS service (*cf.* Fig. 5.2). As all the virtual organizations may use the OSS service, activity pattern discovery and recommendation supports knowledge sharing among VOs operating inside the SOVOBE. Notion of VO and SOVOBE is formalized in Section 5.4.1.

4. Context model

An observation underlying the RMV method is the following: context of an activity instance affects its execution and outcomes, so indirectly it affects the course of VO collaborative process instance execution. The RMV method aims at categorizing all the meaningful information associated with activity instance execution and aggregate them into a context that is later used during activity pattern recommendation.

Formalization of the RMV method context model is presented in Section 5.4.2. In the RMV method, the context of an activity instance is composed of five elements:

1. *Event context attributes* associated with events of an activity instance;
2. *Process instance attributes* which capture static information that refers to a process instance. One obligatory attribute of a completed VO collaborative process instance is the *outcome* of this instance;
3. *A sequence of events* that occurred before the events associated with the activity instance execution (inclusive);
4. *A set of descriptions* of *service consumers and service providers* that were involved in VO collaborative process instance execution before the activity instance (inclusive) was executed;
5. *A set of events* that occurred in other VO collaborative process instances at the time when the activity instance was executed.

5. Activity pattern discovery technique

In the RMV method, particular behavior of collaborators is considered to be an activity pattern if it appeared in a predefined percentage of completed VO collaborative process instances recorded in the VO collaborative process event log. Discovery of activity patterns in the event log includes:

- Discovery of activities and temporal relations among them;
- Discovery of collaborators and service interfaces associated with activities, as well as analysis of their characteristics to extract classes of service entities;
- Discovery of service relations among collaborators associated with activities and analysis of their characteristics to extract service requirements;
- Discovery of activity pattern context.

Activity patterns are discovered in two steps: (1) discovery of sequence patterns (formalized in Section 5.5.1), (2) transformation of sequence patterns into activity patterns (formalized in Section 5.5.2).

In the first step, a modified version of the *PrefixSpan* algorithm (*cf.* Section 4.3) is used in the RMV method to discover sequence patterns. Main two modifications of the *PrefixSpan* algorithm include: (1) redefinition of the notion of prefix and suffix to capture the overlapped relation among activities; (2) discovery of sequence patterns on both the activity and the attribute level.

In the second step, sequence patterns are transformed to activity patterns by parsing sequence pattern attributes to identify frequent activities, service entities, classes of service entities and service requirements.

An activity pattern may appear in many contexts. All the contexts of an activity pattern are discovered together with this activity pattern.

6. Context-aware recommendations

Analysis of activity pattern contexts is crucial in the second phase of the RMV method devoted to generation of recommendations. The aim of this phase is to provide recommendations by finding the best matching between the current context of an activity instance being a part of the running VO collaborative process instance, on the one hand, and discovered contexts of activity patterns, on the other hand. Formalization of the recommendation model is presented in Section 5.4. The recommendation method is formalized in Section 5.5.

A *context class* expresses some constraints concerning the context of an activity pattern that must be met by an activity pattern to be recommended. In the RMV method, identification of a set of activity patterns with contexts matching a given context class is based on *context distance.* The context distance metric indicates difference between a context class and a context. Context analysis during recommendation supports the adaptation of the VO collaborative processes to changing environment. If circumstances of running VO collaborative process instance change, the context class changes and the set of recommended actions changes as well.

The most frequent activity patterns with low context distance metric are validated using statistical metrics of *sensitivity* and *specificity* (Simon & Boring, 1990). Finally, recommended activity pattern is supplemented with two more values: (1) *recom_index* indicates the expected cost of the process instance when following the recommendation, and (2) *nonrecom_index* indicates expected cost of the process instance when not following the recommendation. The most attractive patterns are those with high values of both *sensitivity* and *specificity*, while a small value of *recom_index* in comparison with *nonrecom_index*.

Recommended activity patterns are merged into one *generic activity pattern* capturing all the behavior described by each activity pattern separately. The RMV method encompasses a method of merging activity pattern service oriented summaries of a process models, service network schemas and service networks. An output recommendation consists of a generic activity pattern and a set of activity patterns.

The final decision to follow particular recommended activity pattern is made by a group of selecting collaborators.

7. Activity pattern instantiation

Preferably, discovered activity patterns are on the executable level (*cf.* Section 3.1.3). However, if the variety of VO collaborative process instances is too big, identification of activity patterns on the abstract or prototype levels is also assumed. Abstract and prototype activity patterns miss complete mapping of service entities to classes of service entities and – as a consequence – to activity descriptions. Missing assignments of service entities to classes of service entities are found during *instantiation* of activity patterns. The RMV method supports selecting collaborators through generation and comparison of various assignments of service entities to classes of service entities.

In the RMV method, the problem of service entities selection is transformed to the problem of subgraph search in the SOVOBE social network. Classes of service entities and service requirements from an activity pattern are used as requirements that must be satisfied to the highest possible extent by a set of service entities. The problem is solved using an approach based on a genetic algorithm (Mitchell, 1998). Two functions are used to guide generation of genomes in each iteration of the genetic algorithm: function evaluating satisfaction of a class of service entity by a service entity and function evaluating satisfaction of a set of service requirements by a set of service entities.

Formalization of activity pattern instantiation problem is presented in Sections 5.4.6 and 5.5.4.

8. Parameterization

Parameterization of the RMV method is made by the use of *explicit* and *implicit* collaborator preferences.

Explicit collaborator preferences are defined during method configuration or are provided to OOS within recommendation request. The set of parameters is associated with various parts of the RMV method and directly influence its effectiveness, quality and results. Parameters are used by the Process Miner and Recommendation Manager components from Fig. 5.2. Definition of these parameters must take into account the characteristics of a specific VO collaborative process supported by the RMV method. The parameters specific for various parts of the RMV method are presented in corresponding sections. The summary of parameters in given in Section 5.6.

Implicit collaborator preferences are not provided by the users, but they are calculated by the Recommendation Monitor (*cf.* Fig. 5.2). Implicit collaborator preferences influence the ranking of recommended activity patterns presented to collaborators. There are two metrics:

- *Confidence indicator* – states the willingness of the collaborators to follow recommendations; if the collaborators perform activities that follow the recommended activity pattern, the confidence level of the recommendation is raised, otherwise it is lowered; activity patterns with higher confidence indicators are ranked higher by the Recommendation Manager; the fact of following recommendation is verified using conformance rules generated for the activity pattern;
- *Social coefficient* – indicates the preference of the selecting collaborators toward using recommendations formulated on the basis of VO collaborative process instances they were involved in (the *group* approach) or the preference to follow recommendations formulated on the basis of VO collaborative process instances they did not participate in (the *generic* approach).

5.4. RMV Method Formal Model

5.4.1. Collaborative Process and VO

In this section, the concept of VO collaborative process and virtual organization is formally defined. These notions are necessary to introduce a formal definition of VO collaborative process event log, VO collaborative process context, activity pattern and VO collaborative process instantiation problem.

Definition 5.1. (Collaboration) A *collaboration* arises when two actors alternately and mutually play roles of service consumer and service provider within one process instance.

Definition 5.2. (Collaborators) Actors involved in collaboration are called *collaborators*.

Definition 5.3. (Collaborative Process) A process is *collaborative* if some actors involved in it are collaborators.

Definition 5.4. (Autonomous member) An *autonomous member* is a member that is legally independent and aims at fulfillment of its own goal that may be different from the goal of an organization it is a member of.

Definition 5.5. (Virtual organization) A *virtual organization* vo_i is an organization whose members are autonomous collaborators involved in execution of a collaborative process to achieve vo_i goal, where at least one member is an organization.

Formally, virtual organization vo_i is a quadruple: $\langle \eta_i, M_i, SI_i, R_i \rangle$, where:

- η_i is a collaborative process model being a vo_i plan to achieve its goal;
- M_i, is a set of members of virtual organization vo_i, $|M_i| \geq 2$;
- SI_i is a set of service interface descriptions used by members of virtual organization vo_i, where $SI_i \neq \emptyset$;
- R_i is a set of relations among members of virtual organization vo_i, where $R_i \neq \emptyset$.

Let *VO* denote the set of all the virtual organizations $VO = \{vo\}$.

Definition 5.6. (Virtual Organization Breeding Environment) A *Virtual Organization Breeding Environment (VOBE)* is a virtual organization whose immutable goal is to provide services to create the conditions and environment to support rapid and fluid creation of other virtual organizations.

5.4.2. Collaborative Process Context

In this section, the concept of context in VO collaborative process instances is formally defined. The notion of context is required to define contextual event and contextual trace. The notion of context is also crucial in discovery and recommendation of activity patterns presented in Section 5.4.4 and Section 5.4.5.

Generic context definitions

For the purpose of identification of activity patterns, a special type of attribute name is distinguished called context feature.

Definition 5.7. (Context Feature). A *context feature cf* is a name of an attribute.

Let $CF = \{cf\}$ denote the set of all the context features. Note that $CF \subseteq AN$.

Definition 5.8. (Context Element). An *event context element* ce is an attribute whose name is a context feature cf, $ce = \langle cf, av_{ce} \rangle$.

Let $ce(t)$ denote an event context element captured at time *t*.

Definition 5.9. (Context). A *context* $co(t)$ is an object whose attributes are context elements $ce(t)$.

Let $co(t, cf)$ denote a subset of context elements of context $co(t)$ where context element name is *cf*.

Let $CO^* = \{co\}$ denote the set of all the contexts.

Event context

Definition 5.10. (Event Context). An *event context* $co(e)$ is a context of event e observed at time $e(t)$.

Example 5.1. Event context element is used to describe circumstances in which an event was observed, e.g., current rate USD/PLN, particular state of social relations among collaborators, or weather conditions. An event context element indicating a particular USD/PLN rate on Monday, June 26th 2013 at 3:52 p.m. is:

- $ce(e) = \langle USD/PLN,\ 3.1931\ PLN \rangle$, where $e(t) =$ *Monday, June 26th 2013 at 3:52 p.m.* ▪

Example 5.2. Consider event context object $TypicalStockDay$ captured at time *t* having the following set of context elements:

- $\langle USD/PLN,\ 3.1931\ PLN \rangle$,
- $\langle NASDAQ, 3605.19 \rangle$. ▪

Event context is not specific for a single event. Instead, one event context is shared by many events executed at time *t*.

Definition 5.11. (Event Social Context Element). An *event social context element sce* is an event context element whose name is *SOCIAL* and value is a service network σ, i.e., $sce = \langle SOCIAL, \sigma \rangle$.

Definition 5.12. (Social Event Context). A *social event context* is an event context whose at least one attribute is a social context element.

Activity instance context

Let $SE(e) = \{\langle se, c \rangle\}$ denote a set of descriptions of service consumers and service providers who were involved in process instance execution before the event *e* inclusive, where *se* is a service entity description and *c* is a number of events a particular service entity description appeared in.

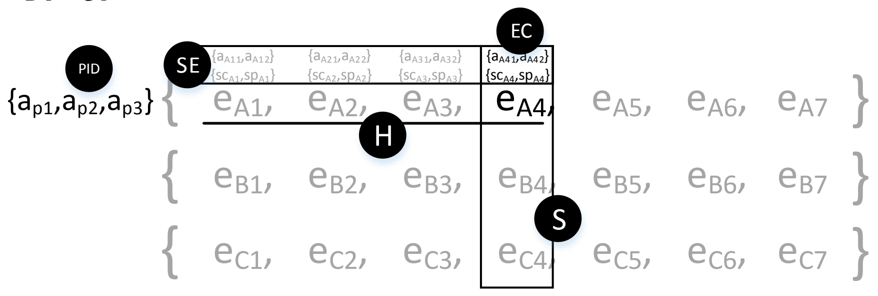
Definition 5.13. (Activity Instance Context). An *activity instance context* $co(vi)$ is a context consisting of the five context elements:

1. $\langle EC, co(e) \rangle$,
2. $\langle H, \tau(p, e) \rangle$,
3. $\langle SE, SE(e) \rangle$,
4. $\langle PID, pid_p \rangle$, and
5. $\langle S, E(e, e(t), (t - \Delta)) \rangle$,

where e is the event from the set $E_p(vi)$ with the highest timestamp and p is a process instance *of* vi. EC, H, SE, PID, S are context element names.

Example 5.3. Consider an event log from Fig. 5.4. Assume that for each vi_i $|E_p(vi_i)| = 1$, $\Delta = 0$. Then, each activity instance was executed by a unique service consumer and service provider. Then context $co(ai_{A4})$ equals:

1. $\langle EC, \{a_{A41}, a_{A42}\} \rangle$,
2. $\langle H, \{e_{A1}, e_{A2}, e_{A3}, e_{A4}\} \rangle$,
3. $\langle SE, \{\langle sc_{A1}, 1 \rangle, \langle sc_{A2}, 1 \rangle, \langle sc_{A3}, 1 \rangle, \langle sc_{A4}, 1 \rangle, \langle sp_{A1}, 1 \rangle, \langle sp_{A2}, 1 \rangle, \langle sp_{A3}, 1 \rangle, \langle sp_{A4}, 1 \rangle\} \rangle$,
4. $\langle PID, \{a_{p1}, a_{p2}, a_{p3}\} \rangle$, and
5. $\langle S, \{e_{B4}, e_{C4}\} \rangle$,



EC
PID
SE
{a_p1,a_p2,a_p3}
{a_A11,a_A12} {sc_A1,sp_A1}
{a_A21,a_A22} {sc_A2,sp_A2}
{a_A31,a_A32} {sc_A3,sp_A3}
{a_A41,a_A42} {sc_A4,sp_A4}
{ e_A1, e_A2, e_A3, e_A4, e_A5, e_A6, e_A7 }
H
{ e_B1, e_B2, e_B3, e_B4, e_B5, e_B6, e_B7 }
S
{ e_C1, e_C2, e_C3, e_C4, e_C5, e_C6, e_C7 }

Fig. 5.4. Four elements of activity instance context

Note that activity instance context is unique within one process instance.

Context class

Definition 5.14. (Context Element Constraint) A *context element constraint* ce^{α} is an attribute constraint $\langle cf, \vartheta_{ce}\rangle$, where cf is a context feature being the name of the context element constraint and ϑ_{ce} is a predicate.

Definition 5.15. (Context Class). A *context class* co^{α} is a class whose each attribute constraint is a context element constraint.

Let $class(co)$ be a user-defined function mapping context co into context class co^{α}.

Definition 5.16. (Social Context Element Constraint). A *social event context element constraint* sce^{α} is an event context element constraint whose name is $SOCIAL$ and predicate is $\oplus\,\sigma^{\alpha}$, where σ^{α} is a service network schema and $\oplus$ is a compliance relation, i.e., $sce^{\alpha} = \langle SOCIAL, \oplus\,\sigma^{\alpha}\rangle$.

Definition 5.17. (Social Context Class). A *social event context class* sco^{α} is a context class whose at least one context element constraint is social context element constraint.

Using event context class it is possible to select a subset of all the event contexts CO being instances of context class co^{α} from CO^{*}.

Example 5.4. Consider an $EconomicStagnation^{\alpha}$ event context class, having the following set of event context element constraints:

- $\langle USD/PLN, > 3.1000\ PLN\rangle$, and
- $\langle NASDAQ, < 3800\rangle$.

An event context $TypicalStockDay$ is an instance of an event context class $EconomicStagnation^{\alpha}$, because:

- $\langle USD/PLN, 3.1931\ PLN\rangle \succ \langle USD/PLN, > 3.1000\ PLN\rangle$, and
- $\langle NASDAQ, 3605.19\rangle \succ \langle NASDAQ, < 3800\rangle$.

Event context $DayInMay = \{\langle USD/PLN, 3.1850\ PLN\rangle, \langle NASDAQ, 3605.19\rangle, \langle Temperature, 30^{o}C\rangle\}$ is also an instance of context class $EconomicStagnation^{\alpha}$. Thus context $DayInMay$ and $TypicalStockDay$ are equal according to context class $EconomicStagnation^{\alpha}$.

Another example concerns event context class $HotDay^{\alpha} = \{\langle Temperature, > 25^{o}C\rangle\}$. Contexts $DayInMay$ and $TypicalStockDay$ are not equal according to context class $HotDay^{\alpha}$. ▪

Instances of classes of activity instance contexts are found in a similar way.

Definition 5.18. (Context Distance). A *context distance* is a user-defined function $eval(co, co^{\alpha})$ that returns a level of compliance of context co to context class co^{α}, where the smaller context distance the more similar is context co to the context class co^{α}.

Example 5.5. Consider an event context $BeforeChristmas$ composed of the following event context elements:

- $\langle USD/PLN, 3.0050\ PLN\rangle$, and
- $\langle NASDAQ, 3605.19\rangle$.

Consider an event context class $EconomicStagnationPlus^{\alpha}$:

- $\langle USD/PLN, > 3.1000\ PLN\rangle$, and
- $\langle NASDAQ, < 3800\rangle$.

Consider a user-defined function $eval$ that assigns weights to each context element constraint, i.e., 0.8 and 0.4. Then, weights assigned to attribute constraints from $EconomicStagnationPlus^{\alpha}$ which are not satisfied by context element from $BeforeChristmas$ are summarized. For context $BeforeChristmas$ and context class $EconomicStagnationPlus^{\alpha}$ only context element constraint $ce = \langle USD/PLN, > 3.1000\rangle$ with weight 0.8 is satisfied. The weight of the second unsatisfied constrain is 0.4. Thus, the value of $eval\left(BeforeChristmas,\ EconomicStagnationPlus^{\alpha}\right) = 0.4$. ▪

In real cases more refined context distance metrics may be used. For instance, when comparing activity instance context with activity instance context class, evaluation of similarity between partial traces from activity instance context feature H (*cf.* Definition 5.13) can take advantage of advanced techniques of partial trace comparison (*cf.* Section 4.3).

5.4.3. Collaborative Process Event Log

Definition 5.19. (Context Event Attribute). A *context event attribute* is an attribute of an event being event context element.

Definition 5.20. (Contextual Event). A *contextual event* is an event having at least one context event attribute.

Definition 5.21. (Service Entity Event Attribute). A *service entity event attribute* is an attribute of an event whose attribute value is service entity description.

Definition 5.22. (Event Social Attribute). An *event social attribute* is an attribute of an event whose attribute value is an object describing the arcs adjacent to service entity description in the service network.

Definition 5.23. (Social Event). A *social event* is an event having at least one service entity event attribute and one social attribute.

Definition 5.24. (Collaborative Event). An event that is contextual and social is called a *collaborative event*.

Collaborative event attribute set includes:

- $e(id)$ is the unique identifier of the event;
- $e(processId)$ is the unique identifier of the VO collaborative process;
- $e(activityId)$ is the unique identifier of the activity description ;
- $e(processInstanceId)$ is the unique identifier of the VO collaborative process instance;
- $e(activityInstanceId)$ is the unique identifier of the activity instance;
- $e(time)$ is the *timestamp* of event *e*;
- $e(trans)$ is the *type of* event *e* (for instance: start, resume, suspend, complete);
- $e(context)$ is the event context attribute;
- $e(SC)$ is a unique identifier of a service consumer;
- $e(SP)$ is a unique identifier of a service provider;

- $e(SI)$ is a unique identifier of a service interface;
- $e(serviceEntity - SC)$ is a service consumer description;
- $e(serviceEntity - SP)$ is a service provider description;
- $e(serviceEntity - SI)$ is a service interface description;
- $e(social - SC)$ is an event social attribute referring to service consumer;
- $e(social - SP)$ is an event social attribute referring to service provider;
- $e(social - SI)$ is an event social attribute referring to service interface.

Note that only event attribute values $e(t)$, $e(context)$ and $e(trans)$ are different for events associated with the same activity instance. It is assumed that other attribute values are the same for all the events associated with one activity instance attribute.

Example 5.6. Consider a collaborative event $SubmitLoanApplication$ from trace recorded during $LoanRequest$ collaborative process instance execution. $SubmitLoanApplication$ event has the following set of attributes:

- $e(id) = 80$,
- $e(processId) = LoadRequest$,
- $e(activityId) = SubmitLoanApplication$,
- $e(processInstanceId) = LoanRequest - 2867$,
- $e(activityInstanceId) = SubmitLoanAplication - 3465$,
- $e(time) = Monday, June, 27th, 12{:}59{:}23{,}123$,
- $e(trans) = complete$,
- $e(context) =$ $\langle DayInMay = \{\langle USD/PLN, 3.1850\ PLN\rangle, \langle NASDAQ, 3605.19\rangle,$ $\langle Temperature, 30^{o}C\rangle\}\rangle$
- $e(SC) = JakubP$,
- $e(SP) = LoanDept$,
- $e(SI) = SubmissionApp$,
- $e(serviceEntity - SC) = \langle JakubP, \{\langle age, 34\rangle\}\rangle$,
- $e(serviceEntity - SP) = \{LoanDept, \{type, Internet\}\}$,
- $e(serviceEntity - SI) = \langle SubmissionApp, \{\langle readyToUse, May\ 1 - 3\rangle\}\rangle$,
- $e(social - SC) = \langle JakubP, \{\langle location, Poznań\rangle\}\rangle$,
- $e(social - SP) = \langle LoanDept, \{\langle location, Poznań\rangle\}\rangle$. ▪

Definition 5.25. (Collaborative Trace). A *collaborative trace* is a trace where each event is collaborative.

5.4.4. Activity Patterns

Temporal graph

Let $e_{s,vi} \in \tau_p$ denote an event corresponding to starting of activity instance *vi* execution.
Let $e_{e,vi} \in \tau_p$ denote an event corresponding to completion of activity instance *vi* execution.
Let $st(vi) = e_{s,vi}(t)$.
Let $et(vi) = e_{e,vi}(t)$.

Definition 5.26. (Relation: *Is followed by*). In a VO collaborative process instance p, an executed activity instance vi_i is *followed* by activity instance vi_j, where $i \neq j$, if $st(vi_j) \geq et(vi_i)$.

Definition 5.27. (Relation: *Are overlapped*). In a collaborative process instance p, two activity instances, vi_i and vi_j, where $i \neq j$, are *overlapping* if $st(vi_j) \leq et(vi_i) < et(vi_j)$ or $st(vi_i) \leq st(vi_j) < et(vi_i)$.

Definition 5.28. (Relation: *Is directly followed*). An executed activity instance vi_i is *directly followed* by activity instance vi_j, where $i \neq j$, in a collaborative process instance p, if vi_i is followed by vi_j and there does not exist a distinct activity instance vi_k in p such that vi_i is followed by vi_k and vi_k is followed by vi_j.

Definition 5.29. (Temporal Graph). *Temporal graph* G_p of a VO collaborative process instance p is a directed acyclic graph $G_p = \langle VID, E \rangle$, where VID is the set of activity instance descriptions, and E is the set of arcs representing *is followed by* and *overlapped* relations and defining partial ordering of activity instance descriptions from VID.

Activity graph

Definition 5.30. (Activity Graph). *Activity graph* Y_p is a prototype service protocol, where the process model is a temporal graph G_p.

Example 5.7. An example of activity graph is presented in Fig. 5.5. ▪

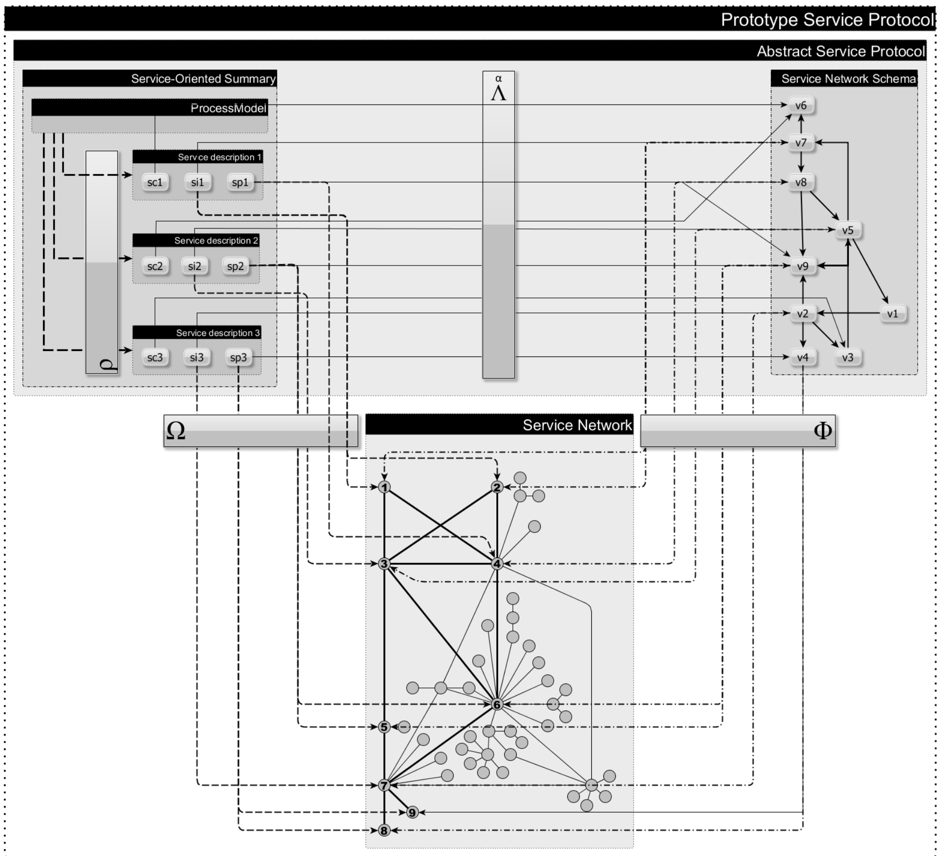


Fig. 5.5. Activity graph

Definition 5.31. (Supported Activity Graph). Activity graph Y' *is supported by* activity graph Y if:

- all followed and overlapped relations that exist in $G_p{}'$ are present in G_p,
- all activities that exist in $G_p{}'$ are present in G_p,
- all service descriptions that exist in Y' are present in Y,
- service network schema of Y' is a sub-network of service network schema of Y,
- relation $\rho_{Y'}$ is a restriction[11] of relation ρ_Y to $\rho_{Y'}$,
- relation $\pi^{\pi}_{Y'}$ is a restriction of relation π^{π}_{Y} to $\pi^{\pi}_{Y'}$,
- relation $\overset{\alpha}{\Lambda}_{Y'}$ is a restriction of relation $\overset{\alpha}{\Lambda}_{Y}$ to $\overset{\alpha}{\Lambda}_{Y'}$,
- relation $\Omega_{Y'}$ is a restriction of relation Ω_Y to $\Omega_{Y'}$,
- relation $\Phi_{Y'}$ is a restriction of relation Φ_Y to $\Phi_{Y'}$,
- set of service entity descriptions assigned to service description elements in Y' is a subset of service entity descriptions assigned to service description elements in Y.

Let $Y' \pm$ Y denote the fact that Y' is supported by Y.

Example 5.8. In Fig. 5.6 two activity graphs are presented. Activity graph Y' on the right is supported by an activity graph Y on the left, i.e., $Y' \pm Y$. Although, temporal graphs in both activity graphs are the same, the number of classes of service entity descriptions from service network schema associated with service descriptions is smaller in Y'. As a consequence, the set of service entity descriptions assigned to elements of service descriptions is a subset of service entity descriptions from Y. Thus, $\overset{\alpha}{\Lambda}$, Ω, Φ relations from Y' are restriction of analogous relations from Y. ▪

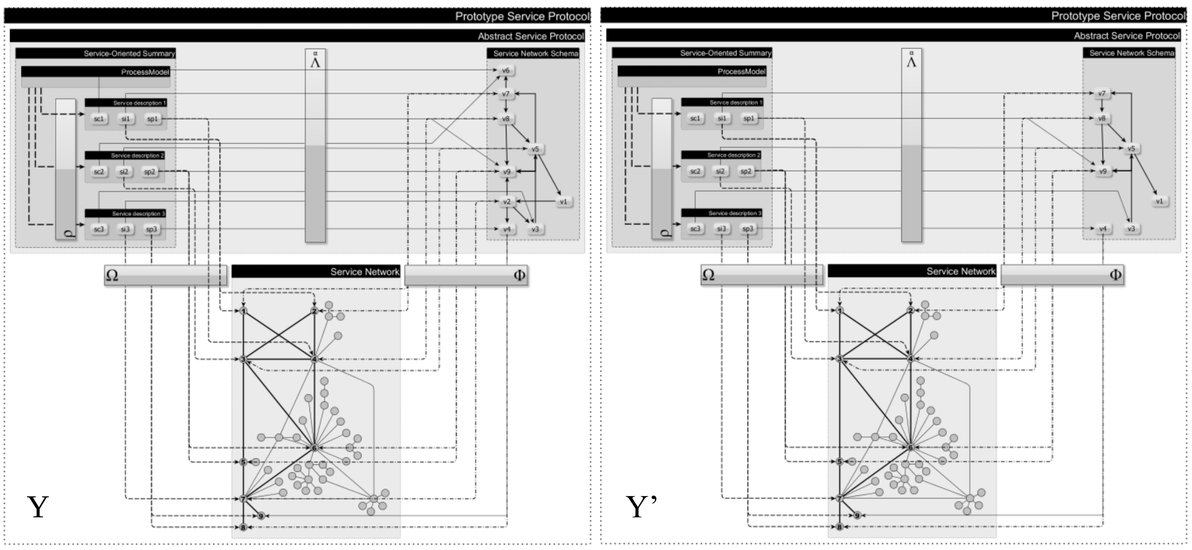


Fig. 5.6. Supporting activity graph

Definition 5.32. (Activity Subgraph). An activity graph Y' is an *activity subgraph* of activity graph Y, where $Y' \neq Y$ if:

- $VID_{A'} \subset VID_A$ and for any pair of vertices $vid_1, vid_2 \in VID_{A'}$, there is an arc in $G_{A'}$ connecting vid_1 to vid_2 if and only if there is an arc in G_A connecting vid_1 to vid_2,

[11] Restriction of function, http://en.wikipedia.org/wiki/Restriction_(mathematics)/

- *Y'* is supported by *Y*.

Note that if *Y'* is a subgraph of *Y* then $Y \pm Y'$.

Example 5.9. Activity graph *Y''* presented in Fig. 5.9 is a subgraph of activity graph *Y'*. The set of activities and relations in *Y''* is a subset of activities and relations in *Y'*, i.e., temporal graph from *Y''* has only two activities and one followed relation, while temporal graph from *Y'* has three activities, two followed relations and one overlapped relation. As a result, the set of service descriptions of *Y''* is a subset of service descriptions in *Y'* as well. As a consequence, the set of service entities assigned to elements of service descriptions is also a subset of analogue set in *Y'*. Relations $\overset{\alpha}{\Lambda}$, Ω, Φ from *Y''* are restriction of relations from *Y'*. Note that *Y''* supports both *Y* and *Y'*. ▪

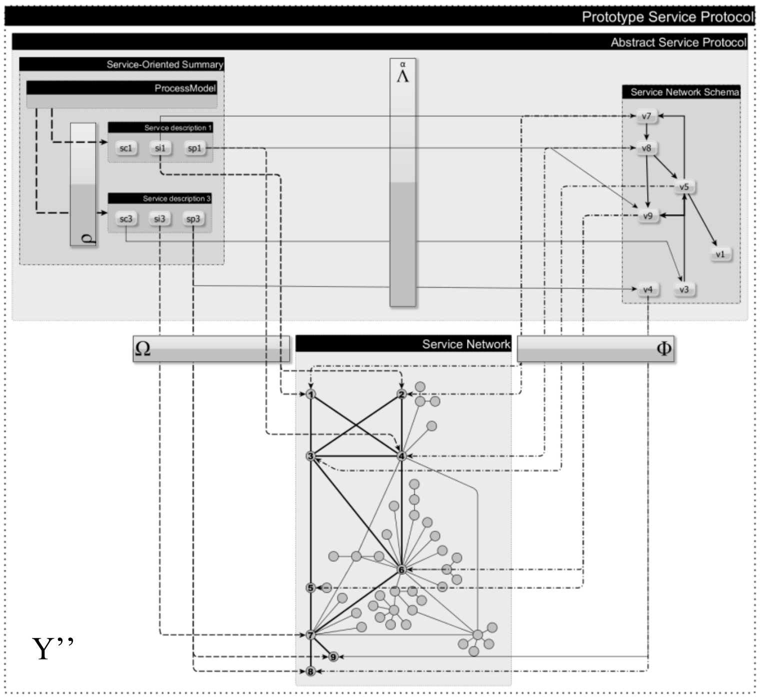


Fig. 5.7. Activity subgraph

Definition 5.33. (Activity Supergraph). If *Y'* is an activity subgraph of *Y*, then *Y* is an *activity supergraph* of *Y'*.

Example 5.10. Activity graph *Y* is a supergraph of activity graph *Y''*. ▪

Activity pattern

Let $PatternSearch\ (\tau(p))$ denote a function that discovers a set of activity graphs from collaborative trace $\tau(p)$.

Definition 5.34. (Activity Graph Supported by Collaborative Trace). An activity graph *Y* is said to be *supported by a collaborative trace* τ(p) if $\exists Y_T \in PatternSearch\ (\tau), Y \pm Y_T$.

Definition 5.35. (Activity Pattern). Let *s*% denote a *minimum support threshold*. Given an event log ℓ, an *activity pattern z* is an activity graph that is supported by *s%* of traces from ℓ.

Let $Z = \{z\}$ denote a set of activity patterns.

Let $sup(z)$ denote a function returning a support of z in event log ℓ.

Example 5.11. Consider a set of ten collaborative process instances, a set of ten corresponding collaborative traces and support threshold 30%. Then, only activity graph that is supported by three collaborative traces is considered an activity pattern. ▪

Definition 5.36. (Closed Activity Pattern). Given a set of activity patterns *Z*. An activity pattern *z* is a *closed activity pattern* in *Z* if there exists no proper activity pattern supergraph *z'* such that *z'* has the same support as *z* in *Z*.

Formally, a set of closed activity patterns $CZ = \left\{z | z \in Z \text{ and } \exists! z' \notin Z, \sup(z) = \sup(z')\right\}$, where function *sup* returns the support of *z* in event log ℓ.

Definition 5.37. (Candidate Activity Pattern). An activity pattern z is said to be *candidate activity pattern* if it is unknown if it is a closed activity pattern.

Activity pattern context

Definition 5.38. (Activity Pattern Context in Collaborative Trace). An *activity pattern context* $co(z)$ *in a collaborative trace* τ is a context of the completed activity instance associated with an event e from τ that directly proceeds the first event associated with the activity instance from the activity pattern.

In other words, activity pattern context describes circumstances that appeared during VO collaborative process instance execution directly before the appearance of an activity pattern.

Example 5.12. In Fig. 5.8 an approach to identification of activity pattern context in the RMV method is presented. Temporal graph is supported by event trace τ. Assume that discovered activity pattern is a subgraph of a temporal graph, i.e., a set of activity instances supporting activity pattern is $\{B, D, E\}$. Activity pattern context is a context of activity instance *A* which is the last completed activity instance whose last event precedes events associated with activity instance *B*. Although, event e_3 directly precedes *B* in the trace, it is associated with uncompleted activity *C*. ▪

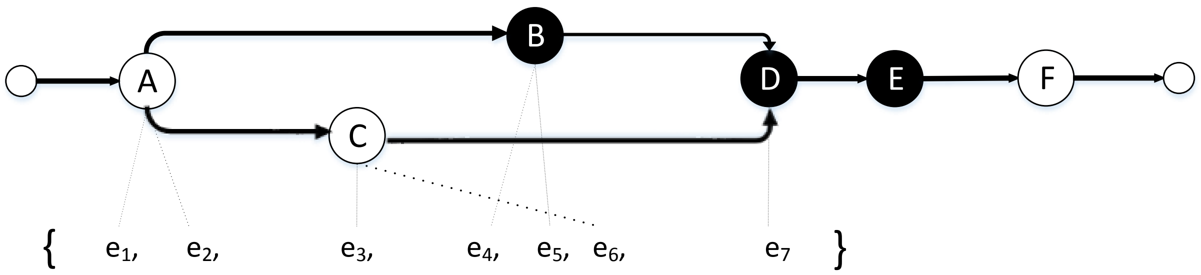


Fig. 5.8. Identification of activity pattern context

Note that one activity pattern supported by *n* traces may have *n* different activity pattern contexts. Moreover, a value of activity pattern context $co(z)$ may not be unique, i.e., more than one activity pattern may be discovered with the same activity pattern context.

Let $fco(z)$ denote a set of pairs $\langle co(z), T\rangle$, where $co(z)$ is an activity pattern context and T is a set of traces supporting activity pattern in context $co(z)$.

Activity pattern discovery problem

Definition 5.39. (Activity Pattern Discovery). An *activity pattern discovery* is a function that maps an event log ℓ onto a set of closed activity patterns, where each activity pattern is supported by at least s% of traces recorded in event log ℓ, and for each activity pattern z a set $fco(z)$ is known.

Let $CMZ = \{z\}$ denote the set of all the closed activity patterns discovered in an event log ℓ for minimum pattern support threshold s.

5.4.5. Recommendations

Let $co(\tau)$ be a context of the activity instance completed as the last one in partial trace τ.

Let $co_{\tau}{}^{\alpha} = class(co(\tau))$.

Definition 5.40. (Target Function). A *target function* $\sigma : L \rightarrow \mathbb{R}$ is a user-defined function which attaches a *target value* to a collaborative trace.

Example 5.13. Examples of target values are: cost or number of involved resources. The target value is calculated on the basis of appropriate event attributes. Event attributes corresponding to activity instance cost can be used to calculate cost of the VO collaborative process instance. ▪

Definition 5.41. (Recommendation index). A *recommendation index recom_index* is the expected target value of the collaborative partial trace τ when τ is continued after performing activity pattern *z*.

A recommendation index for *z* is calculated on the basis of target values calculated for traces supporting *z* in $co_{\tau}{}^{\alpha}$. In other words, by analysis of traces in which an activity pattern was executed in particular contexts, one can make some prediction concerning the outcome of the current collaborative process instance if it is continued after performing activity pattern *z*.

Definition 5.42. (Non-recommendation index). A *non-recommendation index nonrecom_index* is the expected target value when τ is continued after decision of not performing activity pattern *z*.

A non-recommendation index for *z* is calculated on the basis of target values calculated for traces supporting *z* in $co_{\tau}{}^{\alpha}$.

Definition 5.43. (Recommendation Justification). A *justification* $\aleph$ is an object describing recommendation.

Definition 5.44. (Recommendation Element). *Recommendation element* $\delta(\tau, z)$ is a tuple $\langle z, recom\_index, nonrecom\_index, \aleph\rangle$.

Definition 5.45. (Recommendation Query). A *recommendation query* is a tuple $RQ = \{\tau, CMZ\}$.

Definition 5.46. (Recommendation). For a given set of activity patterns *Z*, a *recommendation* $R(\tau)$ be a set of recommendation elements for all the activity patterns from set *CMZ*. Formally, $R(\tau) = \{\delta(\tau, z) | z \in CMZ\}$.

5.4.6. Instantiation

Definition 5.47. (Compliance value). A *compliance value* is the value of the level of compliance of a service network $\sigma = \langle SE, L\rangle$ with a service network schema $\sigma^{\alpha} = \langle SE^{\alpha}, L^{\alpha}, \rangle$ where value is in $\langle 0,1 \rangle$.

Let $compliance(\sigma, \sigma^{\alpha})$ denote a user-defined *compliance* function returning compliance value for a service network σ and service network schema σ^{α}. Then, $\sigma \oplus \sigma^{\alpha} \Leftrightarrow compliance(\sigma, \sigma^{\alpha}) = 1$ and $\sigma \pm \sigma^{\alpha} \Leftrightarrow compliance(\sigma, \sigma^{\alpha}) \in (0,1)$. σ is not compliant with σ^{α}, denoted $\sigma \mp \sigma^{\alpha}$, if $compliance(\sigma, \sigma^{\alpha}) = 0$.

Let σ_{SOVOBE} denote a service network of all the members of SOVOBE.

Definition 5.48. (Activity Pattern Instantiation). An *activity pattern instantiation* is a function that maps a set of service entities from σ_{SOVOBE} onto a set of classes of service entities from service network schema σ^{α} maximizing the value of the $compliance(\sigma_{SOVOBE}, \sigma^{\alpha})$ function.

5.5. RMV Method Partner and Service Selection Technique

In this section the formalization of the main steps of the RMV method is presented. First, the RMV method algorithm of discovery of frequent sequence patterns in behavior of a group of collaborators is formalized (Section 5.5.1). The output of the algorithm is activity sequence patterns. Discovered activity sequence patterns are then analyzed and transformed to activity patterns and then stored in activity pattern repository (Section 5.5.2). Transformation of sequence patterns to activity patterns within the RMV method encompasses identification of service-oriented summary of a process model, service network schema, service network and corresponding mappings. In the third part of this section, the RMV method algorithm for recommendation formulation is formalized (Section 5.5.3). This algorithm aims at recommendation of identified activity patterns to the RMV method users. Finally, RMV method algorithm for activity patterns instantiation is formalized in Section 5.5.4.

5.5.1. Activity Sequence Pattern Discovery

The RMV method algorithm for discovery of frequent sequence patterns in behavior of a group of collaborators presented in this section is based on the modified *PrefixSpan* algorithm (*cf.* Section 4.3). The modification of the classic *PrefixSpan* algorithm is twofold: (1) elaboration of representation of *is followed by* and *overlaps* relations in a form of a sequence of activity instance descriptions, (2) modification of the basic notions used in the algorithm, i.e., the notions of prefix and suffix.

Activity sequences

Definition 5.49. (Activity Instance Description Sequence) An *activity instance description sequence qss* (*activity sequence* in short) is a list of activity instance description sets corresponding to activity instances executed in a VO collaborative process instance. Let $qss = \langle VID \rangle$ denote a sequence, where $VID = \{vid\}$.

A sequence of one-element sets of activity instance descriptions $\langle \{vid_i\}\{vid_j\} \rangle$ denotes that activity instance vi_i *is followed by* activity instance vi_j. Note that square brackets are used for grouping activity instance description sets within the sequence. A two-element set of activity instance descriptions $\{vid_i, vid_j\}$ denotes that activities instance vi_i and vi_j are temporally *overlapped*. The sequence $\langle \{vid_1, vid_2\}\{vid_2, vid_3\} \rangle$ denotes that activity instance vi_1 is followed by vi_3 and activity instance vi_2 overlaps with vi_1 and vi_3. Note that curly brackets are used for grouping activity instance descriptions corresponding to overlapping activity instances.

In this section, for clarity of presentation, the following assumptions are made:

- If all the activity instance descriptions within a sequence are described with a set of attributes having the same names, then only attribute values are presented in an activity sequence;
- If the activity instance description has one attribute, then only the value of the attribute is used in an activity sequence;
- If an activity instance description set has only one element then curly brackets are dropped;
- The requirement for an activity instance description is to have two mandatory attributes with names $activityStart$ and $activityEnd$ is dropped.

Example 5.14. A sequence of two activity instance descriptions having the same set of attribute is: $\langle (A, 23, \otimes), (B, 5, \pm) \rangle$. ▪

Note that round brackets are used for grouping attribute values from one activity instance description.

A set of attributes of activity instance description is defined on the basis of attributes of events generated during execution of an activity instance that this activity instance description represents.

Let $e_{vi_i} \in E_p(vi_i)$ denote an event assigned to activity instance vi_i from the VO collaborative process instance p. Let es_{vi_i} denote a set of attributes of event e_{vi_i}. Then $vid_i = es_{vi_i} \setminus \{e(time),\ e(context),\ e(trans)\}$.

Definition 5.50. (Activity Instance Description Attribute Equality) The activity instance description vid_i *is attribute equal* to vid_j if it is equal according to classifier $\underline{co}$ (*cf.* Definition 2**.5)** where a set of attribute names from $\underline{co}$ is equal to a set of attribute names from vid_i.

Let $vid_i \overset{co}{=} vid_j$ denote that vid_i is attribute equal to vid_j.

Note that if $vid_i \overset{co}{=} vid_j$, then $vid_j \overset{co}{=} vid_i$ is not necessary observed.

Example 5.15. Consider attribute sets of the following activity instance descriptions:

$vid_1 = \{\langle a_1, A\rangle, \langle a_2, 5\rangle, \langle a_3, \otimes\rangle\}$, $vid_2 = \{\langle a_1, A\rangle, \langle a_2, 5\rangle, \langle a_3, \otimes\rangle\langle a_4, \oplus\rangle\}$, $vid_3 = \{\langle a_1, A\rangle, \langle a_2, 10\rangle\}$, $vid_4 = \{\langle a_1, A\rangle, \langle a_2, 5\rangle\}$. Then, $vid_4 \overset{co}{=} vid_1$ but $vid_2 \overset{co}{\neq} vid_1$, $vid_3 \overset{co}{\neq} vid_1$ and $vid_1 \overset{co}{\neq} vid_4$. ▪

Note that the above interpretation of the sequence $\langle\{vid_1, vid_2\}\{vid_2, vid_3\}\rangle$ is different from the one used in majority of classic pattern discovery algorithms including *PrefixSpan*. In the classic approach, the sequence $\langle\{vid_1, vid_2\}\{vid_2, vid_3\}\rangle$ expresses that vi_2 represented by vid_2 in the first set is taking place before the activity instances represented by activity instance descriptions in the second set and activity instance vi_2 represented by activity instance description in the second set occurs after those from the first one. As a consequence of interpretation of activity sequence proposed in the RMV method, in order to support *overlapped* relation among activity instances, the concept of the subsequence must be redefined as compared to the classic approach.

Definition 5.51. (Activity Instance Description Set Containment) An activity instance description set VID_j *contains* VID_i if each activity instance description from VID_i is attribute equal to different attribute instance descriptions from VID_j.

Let $VID_i \overset{co}{\subseteq} VID_j$ denote that VID_j contains VID_i.

Definition 5.52. (Subsequence) A sequence $r = \langle VID_1, VID_2, \dots, VID_n\rangle$ is a *subsequence* of sequence $d = \langle VID_1, VID_2, \dots, VID_n\rangle$ if there exists integers $i_1 < i_2 < \cdots < i_n$ such that $VID_1 \overset{co}{\subseteq} VID'_{i_1}, VID_2 \overset{co}{\subseteq} VID'_{i_2}, \dots, VID_n \overset{co}{\subseteq} VID'_{i_n}$, and there exist no consecutive sets of activity instance description sets VID_j and VID_{j+1} in r such that activity instance description $b \in VID_j$, activity instance description $q \in VID_{j+1}$, and $\{b, q\} \subseteq VID'_{i_j}$ or $\{b, q\} \subseteq VID'_{i_{j+1}}$.

Example 5.16. Consider two sets of activity instance descriptions having one attribute being activity instance identifier. Sequence $r = \langle B, C\rangle$ is a not a subsequence of $d = \langle\{A, B\}\{B, C\}\rangle$. The sequence r denotes *is followed by* relation between B and C, which differs from an *overlapped* relation in d. A subsequence $r' = \langle A, C\rangle$ is a subsequence of sequence d. *r'* corresponds to *is followed by* relation that is also present in d. ▪

Definition 5.53. (Sequence Support) A sequence *d supports* sequence *s* if *s* is a subsequence of *d*. Then *d* is a *super-sequence* of *s*.

Definition 5.54. (Canonical Sequence) A sequence $s = \langle vid_1, vid_2, \dots, vid_n\rangle$ is *canonical* if for each activity instance description set u_j, $1 \leq j \leq m, u_j \not\subset u_{j+1}$, and $u_{j+1} \not\subset u_j$.

Example 5.17. A sequence $\langle A, \{A, B\}\rangle$ is not canonical. In further analysis such sequence is reduced to canonical equivalent$\langle\{A, B\}\rangle$ representing the same behavior. ▪

Activity sequence discovery

Further in this section it is assumed that each activity instance description in an activity sequence consists only of one attribute that is the activity instance identifier. This is only for the sake of clarity of presentation of the proposed concepts. All the statements that are true for one-attribute activity instance descriptions are also true for multi-attribute activity instance descriptions.

The RMV method analyses traces stored in a collaborative event log and transforms information about the events into a non-empty set of activity instance descriptions, where all the *overlapped* and *is followed by* relations are explicitly represented as a sequence.

Definition 5.55. (Sequence Length) Let k denote the *length* of activity sequence qss that is the maximum number of activity instance descriptions connected by *is directly followed* relation in temporal graph G_p corresponding to qss.

Example 5.18. In activity sequence $\langle (A,B)(B,C) \rangle$ the maximum number of activities executed sequentially is two, i.e., activity A and C. Thus, $len(\langle (A,B)(B,C) \rangle) = 2$. ▪

Definition 5.56. (Sequence Size) Let s denote a *sequence size* that is the number of activity instance descriptions in an activity sequence.

Example 5.19. There are three activity instance descriptions in $\langle (A,B)(B,C) \rangle$, i.e., A, B, C. Thus, $size(\langle (A,B)(B,C) \rangle) = 3$. ▪

Each VO collaborative process instance p can be transformed to *l-sequence qss*, where $len(qss) \leq l \leq size(qss)$.

A transformation of a trace τ into a sequence begins with initialization of sets $S = \emptyset$ and $T = \emptyset$. The starting and completion times of activity instances in process instance p are captured in event attribute $e(time)$. Events are traversed in time ascending order. The set T starts to accumulate when the first event with activity instance starting time is visited – activity instance description corresponding to the visited event is added to T. Set T is appended to S when two conditions are satisfied: (1) the completion time of one of activities in T is encountered; (2) at least one new activity instance description has been added to T after T was appended to S. Subsequently, the traverse is continued until the next starting time is visited. At this moment, the subset of activity instance descriptions in T whose completion times appear before the currently visited activity instance description are removed from T since this subset of activity instance descriptions that have appeared in the activity sequence are followed by the current activity instance description. This traversal procedure continues until all the events are visited.

Example 5.20. An example of transformation is presented in Fig. 5.9. First, starting points of activity instances corresponding to activity instance description A and B are visited, $T = \langle A, B \rangle$. Then the completion point of activity instance A is visited and set T is appended to set S, $S = S \cup T = \langle \{A,B\} \rangle$. When starting time of C is visited, A's completion time appears before the C's starting time, A is removed from T, $T = T \backslash \{A\} = \langle B \rangle$. In next steps activities C and D are added to T as their staring points are encountered. At this point $T = \langle B, C, D \rangle$. Then, completion point of B is reached, thus $S = S \cup T = \langle \{A,B\}, \{B,C,D\} \rangle$ and $T = T \backslash \{B\} = \langle C, D \rangle$. When completion times of C and D are reached, T is not appended to S as no new items appeared in T after last appending. When the following E's starting time is visited, T becomes empty because the completion times of B, C, and D have all been traversed. When the last completion time is visited, then $S = S \cup \langle E \rangle = \langle \{A,B\}, \{B,C,D\}, \{E\} \rangle$. ▪

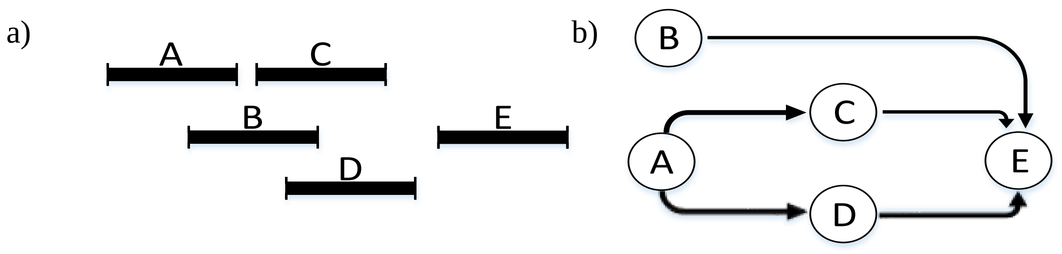


Fig. 5.9. Transformation of a trace to a sequence: (a) activity instances in process instance, (b) corresponding temporal graph

The activity instance descriptions ordering in the same set is unimportant as the activity instances represented by activity instance descriptions from one set were executed in parallel. Thus, it is convenient to assume that all the activity instance descriptions in one set are written with respect to the alphabetical order.

Example 5.21. A sequence should be written in the form of $\langle\{A, B, E\}\{B, C\}\rangle$ instead of $\langle\{B, A, E\}\{C, B\}\rangle$. ▪

Prefix and suffix discovery

In the RMV method, the *PrefixSpan* (*cf.* Section 4.3) is extended in the following way to be able to be applied to identification of activity sequence patterns:

- Redefinition of the notion of prefix and suffix to capture the overlapped relation among activities;
- Identification of sequence patterns on both the activity and the attribute level;
- Technique for generation of frequent 1-sequence patterns for multi-attribute items;
- Incorporating constraint-base analysis to increase usefulness and understanding of generated results;
- Filtering algorithm results in post pruning to include only closed sequence patterns, where closed sequence patterns are those having no super-sequence with the same support.

Without losing generality and for the ease of understanding of the RMV method, modifications introduced to the algorithm are explained under assumption that each activity is described only by its name. All the observations will be later generalized to multi-attribute activities. As each activity has only one attribute, round brackets are dropped.

Consider an activity sequence $s = \langle A, \{B, C, F\}, \{D, F\}, E\rangle$ representing a temporal graph *TG* presented in Fig. 5.10. According to the classic definition, an activity sequence $t = \langle A, F\rangle$ is a prefix of s. Consequently, the suffix created using prefix t is $q' = \langle\{B, C\}, \{D, F\}, E\rangle$. Suffix *q'* corresponds to graph *TG'* in Fig. 5.10. Graphs *TG* and *TG'* are presented in Fig. 5.10. Note that suffix *q'* is incorrect, as *TG'* is not a subgraph of *TG*. The correct suffix is $q'' = \langle\{B, C\}, D, E\rangle$ and corresponding graph *TP''* is also presented in Fig. 5.10. Due to this difference between suffixes *q'* and *q''*, identification of activity patterns using *PrefixSpan* algorithm requires a redefinition of prefix and suffix concepts.

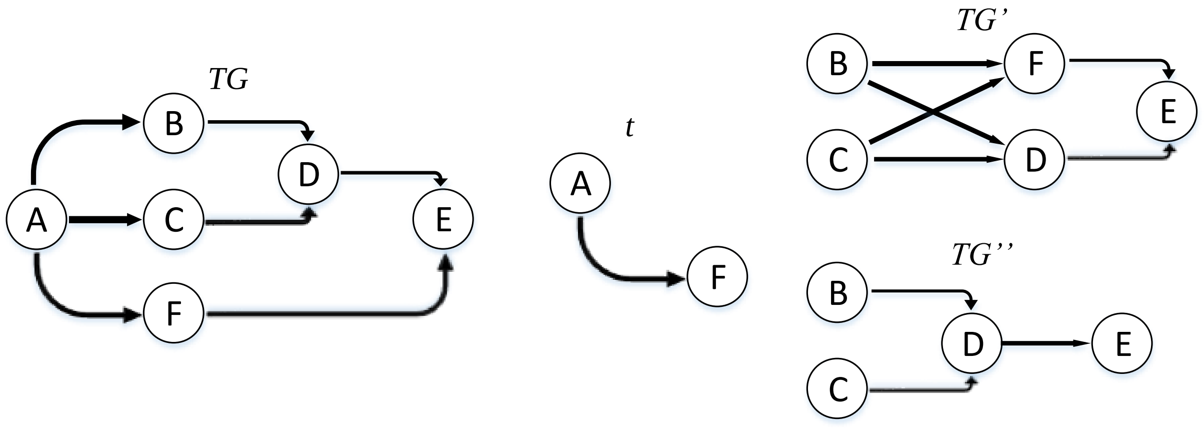


Fig. 5.10. Suffixes of temporal graph *TG*. Incorrect suffix *q'* identified for prefix *t* follows the classic definition and is different from correct suffix *q''*

Definition 5.57. (Prefix) Given an activity sequence $s = \{VID_1, VID_2, \dots, VID_n\}$, then activity sequence $t = \{VID'_1, VID'_2, \dots, VID'_m\}$, where $m < n$, is called the *prefix* of s if t is a subsequence of s for $i \leq m$.

Example 5.22. Activity sequences $\langle A, F\rangle$, $\langle A, \{B, C\}\rangle$ are examples of prefixes of activity sequence $\langle A, \{B, C, F\}, \{D, F\}, E\rangle$, but neither $\langle A, D, F\rangle$ nor $\langle A, E\rangle$ are prefixes. ▪

Note that the above definition of prefix is applicable for multi-attribute activity instance descriptions.

Example 5.23. Sequence $\langle (A, *), (F, \pm)\rangle$ is a prefix of an activity sequence $\langle (A, \oplus, *), \{(C, \oplus, @), (F, \pm, \$)\}\rangle$. ▪

Definition 5.58. (Suffix) Given activity sequence $s = \{VID_1, VID_2, \dots, VID_n\}$, let activity sequence $t = \{VID'_1, VID'_2, \dots, VID'_{m-1}, VID'_m\}$ be a prefix of s, where $m < n$. Activity sequence $q = \{VID''_m, VID''_{m+i}, \dots, VID''_n\}$, where $1 < i < n - m$, is a suffix if and only if two conditions are satisfied: (1) $VID''_m = VID_m \backslash VID'_m$, (2) for each VID_{m+i}, if $VID'_m \subseteq VID$ and $VID''_{m+i-1} \neq VID_{m+i-1}$ then $VID''_{m+i} \neq VID_{m+i} \backslash VID'_m$. Otherwise, for each VID_{m+i}, if $VID'_m \not\subset VID_{m+i}$ then $VID''_{m+i} = VID_{m+i}$.

Let $q = s|t$ denote a suffix of s with regards to prefix t.

Example 5.24. Consider activity sequence $s = \langle A, \{B, C, F\}, \{D, F\}, E\rangle$, and its prefix $t_1 = \langle A, F\rangle$ (*cf.* Fig. 5.11). Then m=2 and n=4. The suffix is initialized as an empty set $q_1 = \emptyset$. First, activity instance description set VID''_2 starting the suffix is identified. Following the rule $VID''_2 = VID_2 \backslash VID'_2$ we have $\{B, C, F\} \backslash F = \{B, C\}$. VID''_2 is then appended to q_1 resulting in $q_1 = \langle \{B, C\}\rangle$. Another element in suffix q_1 is VID''_3. The value of VID''_3 is created on the basis of element $VID''_3 = \{D, F\}$. Because $VID'_2 \subseteq VID_3$, i.e., $F \subseteq \{D, F\}$, and $VID''_2 \neq VID_2$, then $VID''_3 = VID_3 \backslash VID'_2$, i.e., $VID''_3 = \{D, F\} \backslash \{F\} = \{D\}$. Element VID''_3 is appended to q_1, $q_1 = \langle \{B, C\}\rangle \cup \{D\} = \langle \{B, C\}, D\rangle$. Finally, the value of VID''_4 is established. Because $VID'_2 \not\subset VID_4$, then $VID''_4 = \{E\}$ and VID''_4 is appended to q_1, $q_1 = \langle \{B, C\}, D, E\rangle$. Sequence s, prefix t_1, and suffix q_1 are graphically presented in Fig. 5.11. ▪

Example 5.25. Consider again activity sequence $s = \langle A, \{B, C, F\}, \{D, F\}, E\rangle$ and its different prefix $t_2 = \langle A, \{B, C\}\rangle t_2 = \langle A, \{B, C\}\rangle$. Then m=2 and n-4. Element $VID''_2 = VID_2 \backslash VID'_2$, i.e., $VID''_2 = \{B, C, F\} \backslash \{B, C\} = \{F\}$, and VID''_2 is appended to q_2, $q_2 = \langle F\rangle$. Because $VID'_2 \not\subset VID_3$, then $VID''_3 = \{D, F\}$, and VID''_3 is appended to q_2, $q_2 = \langle F, \{D, F\}\rangle$. Finally,

$VID''_4 = \{E\}$ and VID''_4 is appended to t, i.e., $q_2 = \langle F, \{D, F\}, E \rangle$. The resulting suffix q_2 is not canonical thus it is reduced to activity sequence $q_2 = \langle \{D, F\}, E \rangle$. Activity sequence s, prefix t_2, and suffix q_2 are graphically presented in Fig. 5.11. ▪

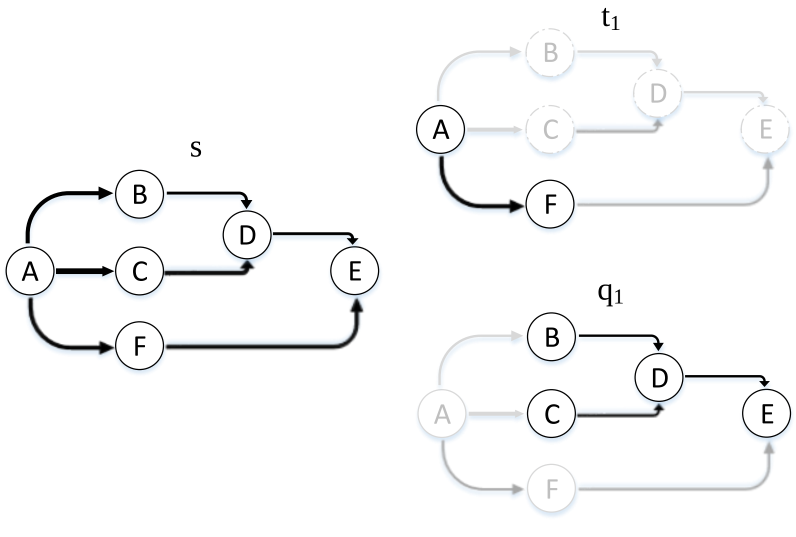


Fig. 5.11. Temporal graph prefixes and corresponding suffixes

Frequent 1-sequences

In the first step of the RMV method algorithm for identification of activity sequence patterns, the database of sequences is scanned to find all the frequent items. Each activity is validated if it is supported by the minimum number of sequences.

Example 5.26. Consider three activity sequences: $s_1 = \langle A, B \rangle$, $s_2 = \langle C, A \rangle$, and $s_3 = \langle C, B \rangle$. Assuming minimum required support equal to 2, a set of 1-element frequent patterns is $qss = \{A, B, C\}$ as all the items appear in at least two activity sequences. ▪

On the contrary, in the RMV method, to identify activity sequence patterns, not only on the level of activity names, also other attributes and their subsets are validated in terms of patterns. As a consequence, it is possible to discover activity sequence patterns with unknown names of some activities. For instance it might not be possible to clearly indicate which activity is performed after activity A. Still it is possible to indicate that the next activity should be performed by John and should last no longer than 30 minutes.

Identification of activity sequence patterns using all the attributes associated with activity instance descriptions requires identification of closed sets of attributes. For this purpose, two steps are performed:

1. First, attributes supported by a minimum number of activity instance descriptions are selected.
2. Second, attributes are used to generate maximum closed sets of attributes; in each generated set, attribute names must be unique; each generated maximum closed set is stored with its support.

Example 5.27. Consider three sequences: $s_1 = \langle (A, 3, *), (B, 3, !) \rangle$, $s_2 = \langle (C, 1, *), (A, 3, !) \rangle$ and $s_3 = \langle (C, 4, *), (B, 5, \#) \rangle$. Assume that the minimum required support is equal to 2. In the first step, the following set of attributes is identified: $sa = \{A{:}\,2,\ B{:}\,2,\ C{:}\,2,\ 3{:}\,2,\ *{:}\,3,\ !{:}\,2\}$, where the notation "$pattern : count$" represents the attribute and its associated support count. A set of 1-element frequent patterns identified in the second step is: $qss = \{(3,!): 2,\ (C,*): 2,\ (A, 3): 2,\ (B): 2\}$. For instance, $(3, !)$ is supported by $(B, 3, !)$ and $(A, 3, !)$. ▪

Constraints

The RMV method approach to identification of activity sequence patterns is based on constraint-based approach to limit the search space and the number of generated results. Following the classic categorization of constraints, the following constraints are used:

- *Duration constraint* – $dur(s) < td$, where td is a given integer; function *dur* calculates the difference between timestamps corresponding to the activities in s completed at the latest, and started at the earliest;
- *Gap constraint* – $gap(s) < tg$, where tg is a given integer; function *gap* calculates the maximum difference between two activities joined by *is followed by* relation;
- *Maximum timestamp constraint* – $timestamp(s) < tm$, where *tm* is a timestamp; function *timestamp* returns the maximum timestamp corresponding to the last activity instance description in sequence *s*.
- *Minimum sequence pattern length constraint* – $len(s) > k$, where k is an integer; function *len* returns the length of *s*.
- *Maximum sequence pattern length constraint* – $len(s) < h$, where h is an integer; function *len* returns the length of *s*.

In the RMV method, every generated activity sequence pattern is validated against prefix constraints before it is used for bi-level projection. The activity sequence pattern that does not satisfy the constraints is not further analyzed in terms of support and is not used for database projection.

Activity pattern support and closed patterns

Having a set of sequences *QSS* and sequence pattern *s*, each *qss* from *QSS* is validated in terms of support for s. Sequences *qss* supports s if: (1) s is a subsequence of *qss*, (2) subsequences *s* meets all prefix anti-monotone.

The pseudo-code for the proposed function *SupportCalc* calculating the support level for pattern *s* is presented in Listing 5.1

Listing 5.1. *SupportCalc* function

```
SupportCalc (
    activity sequence pattern: s,
    sequence set: QSS;
    set of prefix constraints: C): number
{
    sup_s=0;
    QSS'← find all qss in QSS that support s and satisfy constraints from C;

    for each qss from QSS'
    {
            if (s is subsequence of qss and s satisfies C)
            {
                    sup_s++;
            }
    }/* end-for*/
    return sup_s/|QSS|; /* express support in % by dividing number of supporting
                                    sequence by a total number of sequences in database*/
}
```

In the RMV method the closed activity pattern set is created during post-pruning. All the unclosed sequence patterns are filtered out.

Context-aware activity sequence patterns

The RMV method algorithm recursively generates activity sequence patterns and utilizes *SupportCalc* function for verification of their support. The pseudo-code for the algorithm is presented in Listing 5.2 and Listing 3.

The RMV method algorithm presented in Listing 5.2 takes activity sequence database *QQS*, predefined minimum support *s*, and a set of constraints *C* as an input. First, frequent 1-sequences that satisfy a required support level are identified as set *B* (line 7-14). A set of frequent 1-sequences is used to simplify a sequence database and create a pseudo-database, where each sequence consists only of activity instance descriptions that support 1-sequences (line 15). Then pseudo-database and set *B* are used for first database projection (line 16). To improve efficiency of calculations, the algorithm employs the notion of S-matrix tree from the classic *PrefixSpan* algorithm for projecting new partitions, storing sequences and counting their support. As a result of bi-projection, a set of frequent 2-sequence patterns *Z* is created (line 16).

Each activity sequence pattern α from Z is saved with all it contexts (line 19-23). Note that each activity sequence pattern discovered as an extension of activity pattern α will have the same set of contexts. Each sequence pattern α from Z is used for projection of a database (line 24). Then recursion is called for each α and corresponding projected database (line 25). Required support level, set *B* and set of constraints *C* are also always passed to each recursion level.

Listing 5.2. Sequence pattern search initialization

```
0.  PatternSearch (
1.      a set of sequences: QSS,
2.      the minimum support: s,
3.      set of anti-monotone constraints: C): a set of activity patterns
4.  {
5.
6.      frequent ← Ø; /* a set of frequent 1-sequences in QQS*/
7.      B ← find all 1-sequence in QSS;
8.      for each b from B
9.      {
10.             if (SupportCalc(b, QSS, C) not less than s)
11.             {
12.                     frequent ← b;
13.             }/* end-if */
14.     }/* end-for */
15.     QSS ← create pseudo-database using frequent and QSS;
16.     Z ← perform bi-level projection for B, QSS and s;
17.     for each $\alpha$ from Z
18.     {
19.             if ($\alpha$ satisfies all constraints from C)
21.             {
22.                     save pattern $\alpha$ with context;
23.             }
24.             $S|\alpha$ ← build projected database using $\alpha$ and QSS;
25.             $R \leftarrow R \cup$ Recursion($\alpha$, $S|\alpha$, s, B, C);
26.     }/* end-for */
27.   return R;
28.   }
```

The *Recursion* method of the RMV method is presented in Listing 5.3.. First, the list of 1-sequences is updated for particular current pseudo-database *QSS*, i.e., 1-sequences that are not frequent in a pseudo-database are removed (line 7). For each frequent 1-sequence and activity sequence pattern α, a new activity sequence pattern is created and saved if it is frequent and meets constraints *C* (line 8-16). Each saved activity sequence pattern has a size equal to the size of activity sequence pattern α increased by 1. If the length of a saved activity sequence pattern is smaller than maximum allowed length (line 17-18), the next step of the algorithm is performed. A set of frequent 1-sequences is used for database projection (line 20). Each pattern *z* from generated set *Z* is then merged with activity sequence pattern α (line 23). Resulting activity sequence pattern α' has the size of activity sequence pattern α increased by 2. Activity sequence pattern α' is saved if it meets constraints (line 24-27). For each *z* from *Z* a new projection is performed (line 28) and recursion is called (line 29).

Listing 5.3. Sequence pattern search recursion call

```
Recursion (
    activity  sequence pattern: $\alpha$,
    $\alpha$ -projected database: QSS,
    the minimum support: s,
    frequent patterns from previous recursion: B;
    set of anti-monotone constraints: C): a set of activity patterns
{
    B ← update frequent 1-sequence set using B and QSS;
    for (each b from B)
    {
        b' = merge activity pattern $\alpha$ with 1-sequence b;
        if (b' satisfies all constraints from C
                and SupportCalc(b', QSS, C) not less than s)
        {
            save pattern b with context;
        }
    }

    if length of $\alpha$ + length of any element from B is smaller than maximum from C
    {
        Z ← perform bi-level projection using B, QSS and s);
        for each z from Z
        {
            $\alpha'$ = merge activity pattern $\alpha$ with activity pattern z;
            if ($\alpha'$ satisfies all constraints from C)
            {
                save pattern $\alpha'$ with context
            }/* end-if */
            $S|\alpha'$ ← build projected database for z and QSS;
            $R \leftarrow R \cup$ Recursion($\alpha'$, $S|\alpha'$, s, B, C);
        }
}
```

The RMV method algorithm returns a set of closed activity sequence patterns, where each activity sequence pattern is associated with a set of process instance identifiers indicating process instances supporting the pattern. The pseudo-code for the algorithm is presented in Listing 5.4.

Listing 5.4. The first phase of the RMV method algorithm for discovery of activity sequence patterns

```
SequencePatternDiscovery (event log: $L$,
    the minimum support: $s$,
    set of constraints (maximum pattern duration: $td$,
        maximum gap: $tg$, maximum timestamp: $tm$, minimum length: $h$): $C$, ):
                                        a set of closed activity patterns
{
    $QSS \leftarrow \emptyset$;  /* $QSS$ contains a set of sequences */
    $CQSS \leftarrow \emptyset$; /* $CQSS$ contains a set of closed sequence patterns */
    for each trace $\tau$ in $L$
    {
        $QSS \leftarrow QSS \cup$ sequence generated for $\tau$;
    } /* end-for */
    $R \leftarrow$ PatternSearch ($QSS$, $s$, $C$);
    $R \leftarrow$ filter out all the sequences that appear in $CMZ$ that are
                not closed and satisfy constraints from $C$;
    return $R$;
}
```

5.5.2. Activity Pattern Identification

In the RMV method, transformation of an activity sequence pattern to an activity pattern is based on a subset of attributes associated with activity instance description from sequence patterns. The considered set of attributes is: $e(SC)$, $e(SI)$, $e(SP)$, $e(serviceEntity - SC)$, $e(serviceEntity - SI)$, $e(serviceEntity - SP)$, $e(social - SC)$, $e(social - SI)$, $e(social - SP)$ (*cf.* Section 5.4.3).

The following steps are performed in the RMV method when transforming an activity sequence pattern to an activity pattern:

1. Identification of service oriented summary of a process model, in particular process model $\pi^{\alpha} = (VD, E)$;
2. Identification of service network schema $\sigma^{\alpha} = \langle SE^{\alpha}, L^{\alpha} \rangle$ elements, i.e., identification of classes of service entity descriptions from SE^{α} and identification of service requirements from L^{α};
3. Definition of function $\overset{\alpha}{\Lambda} : \left(S^{*\alpha} = SC^{\alpha} \cup SI^{\alpha} \cup SP^{\alpha}\right) \times SE^{\alpha}$ mapping classes of service entity descriptions from the service network schema to elements of service descriptions from a service-oriented summary of a process model;
4. Identification of service network $\sigma = \langle SE, L \rangle$, in particular identification of service entity descriptions from SE and identification of relations among service entity descriptions from L;
5. Definition of function $\Phi : SE \times SE^{\alpha}$ mapping service entity descriptions from service network σ to classed of service entities in service network schema σ^{α};
6. Definition of function $\Omega : E \times S^{*\alpha}$ mapping service entity descriptions from service network σ to elements of service descriptions.

Service-oriented summary of a process model

The RMV method algorithm for identification of a service-oriented summary of a process model is presented in Listing 5.5. The algorithm travers all the activity instance descriptions from activity sequence pattern s (line 5). For each activity instance description vid, a process model activity *t* is created (line 7) having the name equal to an attribute $e(activityId)$ of vid. Note that for some visited activity instance descriptions, $e(activityId)$ may not be set. Then an activity with no name is created in the process model. Created process model activity *t* is added to the set of activities *VD* (line 8). Also the mapping *m* between the activity instance description form the activity sequence pattern and the created activity is kept (line 9). Once all the process model activities are identified, a set of temporal relations *E* is created (line 11) by parsing activity sequence pattern *s*. Interpretation of activity sequence pattern notation in terms of temporal relationships is presented in Section 5.5.1. In lines 12-15, a process model is created together with full service-oriented summary of the process model, i.e., set S_d and mapping ρ^{α} are generated automatically – each activity from the process model has assigned one unique service description.

Listing 5.5. Create service oriented summary of a process model process model π^{α}

```
0.  CreateServiceOrientedSummaryOfProcessModel (
1.      activity  sequence pattern: s): process model
2.  {
3.      m← Ø /* holds mapping of activity instance description to task */
4.      VD← Ø /* holds a set of process model activities t */
5.      for each activity instance description vid form s
6.      {
7.              t = create task (vid);
8.              VD ← t;
9.              m ← add activity instance description vid to task t mapping;
10.     }
11. E ← capture temporal relations among tasks ($\pi^{\alpha}$, s);
12. $\pi^{\alpha} = \langle VD, E \rangle$ ;
13. $\pi^{\alpha}$← m;
14. $\pi_{sos}$← $\pi^{\alpha}$;
15. $\pi_{sos}$ ← create other elements of $\pi_{sos}$;
16. return $\pi_{sos}$;
17. }
```

Service network schema

Identification of a service network schema in the RMV method requires two steps (Listing 5.6): (1) identification of classes of service entities, and (2) identification of service requirements.

Classes of service entities are separately identified for each element type of service description – service consumer, service interface, and service provider (line 6, 14). Given a particular element type of service description *sde*, each activity instance description *vid* in the activity sequence pattern s is visited. For each *vid,* a set of attributes *atts* describing a service entity is identified – depending on the element type of service description, it is described by $e(serviceEntity - SC)$, $e(serviceEntity - SI)$ or $e(serviceEntity - SP)$ attributes, respectively (line 18). Attributes $atts$ are used to create a class of a service entity description se^{α} (line 21). The class is added to the set of all the classes of service entities (SE^{α}). Two mapping are stored: (1) mapping of a class to element of service description (line 22), and (2) mapping of a class to activity instance description vid (line 23). A class is not created if a set of attributes $atts$ is empty, unless analyzed vid has an attribute identifying a service entity of type sde (one of attributes $e(SC)$, $e(SP)$, $e(SI)$ depending on a type of sde) (line 19). At this point, mapping between elements of service descriptions and classes of service entities is 1:1. Next, classes of service entities from SE^{α} are compared with each other. If two classes are exactly the same, i.e., if they have the same set of attributes, then they are merged into one class that is mapped to two elements of service descriptions (line 28). Final mapping of each element of service description to a class of service entity description is in general M:1, i.e., many elements of a service description may be mapped to one class of a service entity.

Similarly to classes of service entities, service requirements are separately identified for each element type of service description (line 6, 14). It starts with identification of a set of social attributes from all the activity instance descriptions from s. Each attribute is stored in the *MATTS* set together with information about activity instance descriptions supporting this attribute (line 34-45). For each attribute *att* (line 36), all the activity instance descriptions *vids* supporting it are visited (line 38-40). For each visited activity instance description $source \in vids$ (line 40), a corresponding class of service entity description $sClass$ of a type sde (line 42-43) is searched for. If for activity instance description *source* no class of service entity of type sde has been identified, i.e., $sClass$ is null, then an empty class is created (line 46-47). In the next step of the algorithm, the next activity instance description is taken from *vids* that is different than *source* (line 49-50) and the corresponding class of service entity description $tClass$ is searched for (line 52-53). Similarly, if $tClass$ is null (line 52), an empty class is created (line 56-57). When both $sClass$ and $tClass$ are not null, a service requirement joining these two classes is created, and the attribute *att* is added to its description (line 54-56). If a service requirement joining $sClass$ and $tClass$ already exists, attribute *att* is added to the existing service requirement. Finally, service network schema is created (line 66). In the final step, for each pair of classes of service interfaces and service providers assigned to the same service description, an implicit "provide/is provided" service requirement is created. This service requirement states that potential service interface description assigned to the class of service interface description must be associated with a service interface provided by an actor being an instance of a class of service provider description assigned to the same service description.

Listing 5.6. Creating service network schema σ^{α} and mapping relation $\overset{\alpha}{\Lambda}$

```
***CreateServiceNetworkSchema***(
    service-oriented summary of process model: $\pi_{sos}$,
    activity sequence pattern: *s*,
    service description elements to class of service description mapper: $\overset{\alpha}{\Lambda}$,
): service network schema
{
    *SDES* ← { *SC*, *SI*, *SP* };    /* create a set of service description element types */
    $SE^{\alpha}$← Ø ;                /* holds classes of service entities */
    $L^{\alpha}$← Ø;                  /* holds service requirements */
    *smapper* ← Ø;                    /* assignment of classes of service entities
                                            to activity instance descriptions */

    /* ###### ***step 1 - create classes of service entities*** ##### */

    for each *sde* from *SDES*        /* for each type of service description element */
    {
        for each activity instance description *vid* form *s*
        {
            *atts* = get service entity attributes (*vid*, *sde*) ;
            if (*atts* is not null or (*atts* is null and entity name of type *sde* is known))
            {
                $se^{\alpha}$ = create class of service entity (*atts*);
                $SE^{\alpha}$← $se^{\alpha}$;
                $\overset{\alpha}{\Lambda}$ ← add mapping (*sde*, $se^{\alpha}$, *vid*, $\pi_{sos}$);
                *smapper* ← add mapping ($se^{\alpha}$, *vid*);
            }/* end-if */
        }/* end-for */
    }/* end-for */
    find and merge common classes($SE^{\alpha}$);

    /* ###### ***step 2 - create service requirements*** ##### */

    for each *sde* from *SDES*
    {
        *MATTS* = get mapping of unique social attributes to
                                activity instance descriptions (*s*, *sde*) ;
        for each *matt* form *MATTS*
        {
            *att* = get social attribute from *matt*;
            *vids* = get activity instance description associated with *att* in *matt*
            for each activity instance description *source* from *vids*
            {
                *sClass* = get class of service entity description
```

```
                                      (*smapper, source*, $\pi_{sos}$, *sde*);
                if *sClass* is null
                {
                    *sClass* = new class of entity description;
                    $\overset{\alpha}{\Lambda}$ ← add mapping (*sde*, *sClass*, *source*, $\pi_{sos}$);
                }/* end-if */
                for each activity instance description *target*
                                      from *vids* different that *source*
                {
                    *tClass* = get class of service entity description
                                      (*smapper, target*, $\pi_{sos}$, *sde*);
                    if *tClass* is null
                    {
                        *tClass* = new class of entity description;
                        $\overset{\alpha}{\Lambda}$ ← add mapping (*sde*, *tClass*, *target*, $\pi_{sos}$);
                    }/* end-if */
                    $l^{\alpha}$= create service requirement (*sClass*,
                                      *tClass*, *att*, *source*, *target*);
                    $L^{\alpha}$← $l^{\alpha}$; /* add to service network schema */
                }/* end-for */
        }/* end-for */
    }/* end-for */
    find and merge common service requirements despite type ($L^{\alpha}$);
    $\sigma^{\alpha} = \langle SE^{\alpha}, L^{\alpha} \rangle$;      /* create service network schema */
    create implicit service requirements($\overset{\alpha}{\Lambda}$, $\sigma^{\alpha}$);
    return $\sigma^{\alpha}$;
}
```

Service network

Identification of a service network in the RMV method is presented in Listing 5.6. Service network is identified in a similar way to identification of a service network schema. There are three main differences: instead of classes of service entities and service requirements, service entities and service relations are created, (2) depending on the type of service entity, one of attributes $e(SC)$, $e(SI)$, $e(SP)$ is used as the name of created service entity, (3) no empty services entities are created. During execution of the algorithm of Listing 5.7, also mapping Φ of classes of service entity to service descriptions is identified (line 18).

Listing 5.7. Creating service network σ

```
CreateServiceNetwork (
    activity  sequence pattern: $s$,
    mapping of classes of service entity descriptions to service entity descriptions: $\Phi$,
    service network schema: $\sigma^{\alpha}$,
    service-oriented summary of a process model: $\pi_{sos}$
): service network
{
    sdes ← { SC, SI, SP};          /* create a set of service description element types */
    $SE$ ← Ø ;                       /* holds service entities */
    $L$ ← Ø;                          /* holds service relations */

    /* ###### create service entities ##### */

    for each sde from sdes          /* for each type of service description element */
    {
        for each activity instance description vid form s
        {
            name = get service entity name (vid, sde) ;
            if name is not null
            {
                se = create service entity (name);
                $\Phi$ ← add mapping (sde, se, $\pi_{sos}$);
                smapper ← add mapping (se, vid);
                $SE$ ←se;        /* add to service network schema */
            }
        }/* end-for */
    }/* end-for */

    /* ###### create service relations ##### */

    for each sde from sdes          /* for each type of service description element */
    {
        matts = get mapping of unique social attribute to
                                        activity instance descriptions (s, sde) ;
        for each matt form matts
        {
            att = get social attribute from matt;
            vids = get activity instance description associated with att in matt
            for each activity instance description source from vids
            {
                sourceEntity = get service entity description
                                                (smapper, source, sde);
                if (sourceEntity is null) continue to next source;
                for each activity instance description target
                                                from vids different that source
```

```
                    {
                            $targetEntity$ = get service entity description
                                                    ($smapper$, $target$, $sde$);
                            if ($targetEntity$ is null) continue to next $target$;
                            for each $targetClass$ from $targetClasses$
                            {
                                    $l$ = create service relation ($sourceEntity$,
                                                    $targetEntity$, $att$, $source$, $target$);
                                    $L \leftarrow l$; /* add to service network schema */
                            }
                    }
            }
        }
    }
    $\sigma = \langle SE, L\rangle$;        /* create service network */
    return $\sigma$;  }
```

Activity pattern

Finally, the algorithm presented in Listing 5.8, shows the ordering of all the operations required to create an activity pattern in the RMV method. First, service-oriented summary of a process model is created based on activity sequence pattern (line 6). In line 7, a service network schema is created basing on the service-oriented summary of the process model π_{sos} and activity sequence pattern s. Basing on activity sequence pattern s and service network schema σ^{α} (line 8), service network σ is created (line 9). As the last element, mapping Ω is identified (line 10-11). Finally all the elements, i.e., π_{sos}, $\overset{\alpha}{\Lambda}$, σ^{α}, Φ, σ, are used for creation of activity pattern z (line 12).

Listing 5.8. Creating activity pattern z

```
0.  CreateActivityPattern (
1.      activity sequence pattern: $s$): activity pattern
2.  {
3.      $\overset{\alpha}{\Lambda} \leftarrow \emptyset$; /* assignment of service description elements to
4.                                              class of service entity */
5.      $\Phi \leftarrow \emptyset$; /* create mapping of classes of service entities to
6.                                              service entity description */
7.      $\pi_{sos}$ = CreateServiceOrientedSummaryOfProcessModel ($s$);
8.      $\sigma^{\alpha}$ = CreateServiceNetworkSchema ($\pi_{sos}$, $s$, $\overset{\alpha}{\Lambda}$);
9.      $\sigma$ = CreateServiceNetwork ($s$, $\Phi$, $\sigma^{\alpha}$);
10.     $\Omega$ = create mapping of elements of service descriptions to
11.                                             service entity descriptions ($\Phi$, $\overset{\alpha}{\Lambda}$);
12.     $z$ = create activity pattern($\pi_{sos}$, $\overset{\alpha}{\Lambda}$, $\sigma^{\alpha}$, $\Phi$, $\sigma$, $\Omega$);
13.     return z;  }
```

Example 5.28. Consider an activity sequence pattern $s = \langle A, \{B, C\}, D \rangle$, where each activity instance description is associated with a set of attributes. Attributes associated with each activity instance description are presented in Tab. 5.2, Tab. 5.3 and Tab. 5.2. In each table, attributes are arranged by activity instance description (*A*, *B*, *C*, *D*) and the type of element of service description – *SC, SI, SP*.

Tab. 5.2 presents values of $e(SC)$, $e(SI)$, $e(SP)$ attributes of activity instance descriptions. Values in the table indicate names of service entities discovered in the activity pattern. For instance, according to the activity pattern, activity A is performed by James, who uses the $PostApp$ service interface provided by the $Loan\ Department$. Note that some activities miss assigned service entities, e.g., there is no service consumer indicated for activity *B*.

Tab. 5.2. Service entities discovered in an activity pattern

#	A	(B	C)	D
SC	<Name, James>			<Name, Mark>
SI	<Name, PostApp>	<Name, SubApp>	<Name, Review>	
SP	<Name, LoanDept>	<Name, LoanDept>	<Name, Mark>	<Name, ClientSupportDept>

In Tab. 5.3, attributes describing service entities typically involved in the execution of an activity are presented. Those are values of $e(serviceEntity - SC)$, $e(serviceEntity - SP)$ or $e(serviceEntity - SI)$ attributes of each activity from the activity pattern. For example, attributes discovered in the activity pattern for activity instance description A which correspond to service consumer description are: {<Profession, Architect>, and <Nationality, Polish>}. Attributes associated with service interface for the same activity instance description are: {<Availability, 24/7>, <MultipleAccess, True>}. Note that if a service entity for a given activity instance description is known (Tab. 5.2), attributes in Tab. 5.3 describe this service entity. In case when a service entity is unknown, the set of attributes includes attributes common for the few service entities. Note that some elements of the service description have no attributes in both tables, e.g., the service consumer description associated with activity instance description *B* has no attributes both in Tab. 5.2 and Tab. 5.3.

Tab. 5.3. Example of service entity attributes discovered in an activity pattern

#service Entity	A	(B	C)	D
SC	<Profession, Architect> <Nationality, Polish>		<Domain, Software> <Experience, High>	<Nationality, German> <Profession, Programmer >
SI	<Availability, 24/7> <MultipleAccess, True>	<Availability, 24/7> <MultipleAccess, True>		
SP	<Size, Big> <Teams, 7>	<Size, Big> <Teams, 7>	<Nationality, German> <Profession, Programmer>	

In Tab. 5.4, social attributes of activity instance descriptions are presented. Those are values of $e(social - SC)$, $e(social - SI)$ or $e(social - SP)$ attributes of each activity instance description from the activity pattern. For example, attributes discovered in the activity pattern for activity instance description A which correspond to service consumer description, indicate that he is from Poznań, he is a member of IEEE and he is categorized as the second-year client. He shares some characteristics with a service provider of the same activity who is also from Poznań.

Tab. 5.4. Example of social attributes discovered in an activity pattern

#social	A	(B	C)	D
SC	<Loc, Poznań> <Membership, IEEE> <ClientAge, 2 years>		<TrustLevel, 10> <Security, High> <Loc, Gdańsk>	<Country, Poland>
SI	<Loc, Gdańsk> <Exp, 2 years> <TrustLevel, 10>	<Loc, Gdańsk> <Exp, 2 years> <TrustLevel, 10>		<Security, High>
SP	<Loc, Poznań> <Org, MB> <Country, Poland>	<Loc, Poznań> <Org, MB> <Country, Poland>	<Country, Poland>	<Org, MB>

In the first step, the service oriented summary of the process model is created basing only on information concerning the set of activity instance descriptions in the activity pattern. Process model activities A, B, C, and D are created together with service descriptions $s_d \in S_d$ and with five temporal relations among them: A proceeds B, A proceeds C, C proceeds D, B proceeds D, B and C are overlapping. The mapping function $\rho: V \to S_d$ is generated automatically.

In the second step of the RMV algorithm, the social network schema is created. Eight classes of service entities are identified that have non-empty set of attributes. The number of classes is reduced in the next step to 5, as some identical classes are merged. Moreover, two classes with an empty set of attributes are identified for elements of service descriptions with known service entities. All the classes are presented in Tab. 5.5. Just for clarity of further discussion, each class is assigned a name. An example of a merged class is class B^α which is common for service interface description of activity A (SI_A) and service interface description of activity B (SI_B). Other merged classes are C^α and E^α. Empty classes are F^α and G^α associated with SP_D and SI_C, respectively. Note that there is no SC_B element of the service description in Tab. 5.5. There is no service entity and no service entity attributes discovered for service consumer of activity B in the activity pattern. Thus, a class of service entity for this element is not created.

Tab. 5.5. Identified classes of service entity descriptions SE^α
and their mapping $\overset{\alpha}{\Lambda}$ to elements of service descriptions

Class name	Attribute set	Support
A^α	{<Profession, Architect>, <Nationality, Polish>}	SC_A
B^α	{<Availability, 24/7>, <MultipleAccess, True>}	SI_A, SI_B
C^α	{<Size, Big>, <Teams, 7>}	SP_A, SP_B
D^α	{<Domain, Software>, <Experience, High>}	SC_C
E^α	{<Nationality, German>, <Profession, Programmer >}	SP_C, SC_D
F^α	$\{\varnothing\}$	SP_D
G^α	$\{\varnothing\}$	SI_C

Data from Tab. 5.4 are used for identification of service requirements which are presented in Tab. 5.6. Just for clarity of further discussion, each service requirement is assigned a name. Note that one additional class of service entity was created in this phase, namely, H^α. This is a consequence of lines 44-48 and 54-48 in Listing 5.6. In case when there is an identified service requirement between two elements of service description having no assigned class of service entity, an empty class of service entity is created and assigned to the proper element of a service description. Such class is used in the definition of service requirement. In this case, empty class H^{α} was created and assigned to SI_D. Its interpretation is that there are no requirements concerning a service interface used in execution of *D,* but such a service entity must provide a high level of security together with a service consumer of activity *C*. The final number of identified classes of service entities is eight.

Tab. 5.6. Identified service requirements L^{α}

Service requirement name	Classes	Attribute set
M^{α}	A^{α}, C^{α}	<Loc, Poznań>, <Country, Poland>
N^{α}	B^{α}, D^{α}	<Loc, Gdańsk>, <TrustLevel, 10>
O^{α}	D^{α}, H^{α}	< Security, High>
P^{α}	C^{α}, E^{α}	<Country, Poland>
R^{α}	C^{α}, F^{α}	<Org, MB>

Note that some social attributes do not appear in any of service requirements, i.e., attributes <ClientAge, 2 years>, <Membership, 10> and <TarustLevel, 10> are not shared by classes of service entities. Thus, they are not translated into service requirements.

In the final step of service network schema identification, implicit relations are created. Classes corresponding to service provider description and service interface descriptions are considered. Identified set of implicit relations is presented in Tab. 5.**7**.

Tab. 5.7. Implicit requirements denoted from $\overset{\alpha}{\Lambda}$ and added to L^{α}

Service requirement name	SI	SP
SP_1^{α}	B^{α}	C^{α}
SP_2^{α}	G^{α}	E^{α}
SP_3^{α}	H^{α}	F^{α}

In the next step of the RMV algorithm, each service entity captured in the activity sequence pattern (Tab. 5.2) is assigned to at least one class of service entity. Discovered assignment of service entities to classes is presented in Tab. 5.8. Note that classes of service entities D^{α} and H^{α} are left without a service entity assigned.

Tab. 5.8. Assignment of classes of service entity descriptions from SE^{α} to service entities from *SE* - mapping relation Φ

Class name	Service entity
A^{α}	James
B^{α}	PostApp, SubApp
C^{α}	LoanDept
E^{α}	Mark
F^{α}	ClientSupportDept
G^{α}	ReviewApp

Finally, the service network is filled-in with service relations. A set of service relations presented in Tab. 5.9 is very similar to a set of service requirements from Tab. 5.6, i.e., majority of service requirements has a corresponding service relation. For instance, service relation M joins service entities assigned to classes of service entities connected by service requirement M^{α}. There are two exceptions form this rule: (1) service relation corresponding to service requirement O^{α} is missing as a consequence of a missing assignment of service entity to D^{α}, (2) relation corresponding to implicit relation SP_3^{α} is missing as a consequence of a missing assignment of service entity to H^{α}.

Tab. 5.9. Identified service relations *L*

Service relation name	Attribute set	Service entities
M	<Loc, Poznań>, <Membership, IEEE>	James, LoanDept
N	<Loc, Gdańsk>, <TrustLevel, 10>	PostApp, SubApp
P	<Country, Poland>	LoanDept, Mark
R	<Org, MB>	LoanDept, ClientSupportDept
SP_1		PostApp, LoanDept
SP_2	<Org, MB>	Review, Mark

Having mapping $\overset{\alpha}{\Lambda}$ (Tab. 5.5) and Φ (Tab. 5.8) it is possible to denote mapping Ω presented in Tab. 5.10. Some elements of service descriptions miss assignment of service entities. Thus the identified activity pattern is not executable (*cf.* Section 3.1.3). Formally, the following condition is not met: $\forall s *^{\alpha} \in S *^{\alpha}, \exists e \in E$ such that $e \, \Omega \, s *^{\alpha}$ and the induced relation $\Phi^{+}: E \times E^{\alpha,+}$is in compliance relation (*cf.* Definition A.11).

Tab. 5.10. Assignment of service entities to elements of service descriptions – mapping **Ω**

Element of service description	Service entity
SC_A	James
SI_A	SubApp, PostApp
SP_A	LoanDept
SC_B	
SI_B	SubApp, PostApp
SP_B	LoanDept
SC_C	
SI_C	Review
SP_C	Mark
SC_D	Mark
SI_D	
SP_D	ClientSupportDept

Graphical representation of the activity pattern discovered by the use of the RMV method is presented in Fig. 5.12. ▪

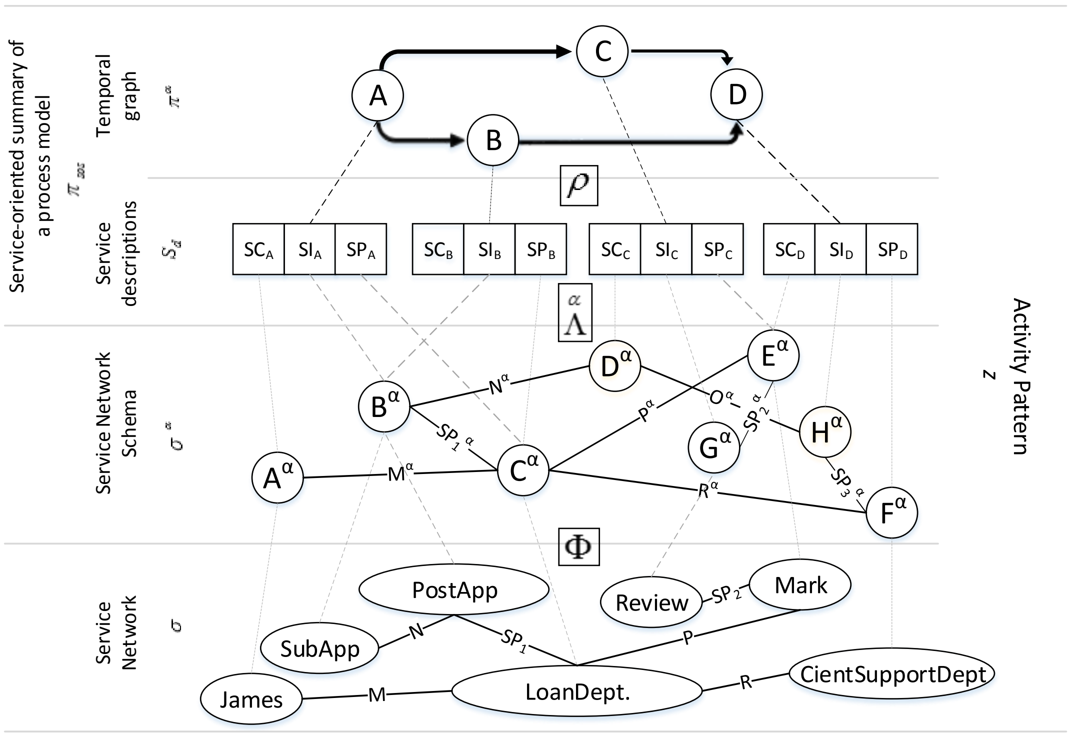


Fig. 5.12. Discovered activity pattern *z*

5.5.3. Recommendation Formulation

Selection of an activity pattern matching a particular context class is transformed to the classification problem. In the RMV method, first a set of activity pattern contexts that are the most similar to a given context class is selected. Then, activity patterns associated with those contexts are evaluated for suitability for recommendation. The RMV algorithm for recommendation formulation includes the following steps:

1) *Enabled contexts* – identification of the most similar contexts based on context distance,
2) *Enabled activity patterns* – identification of activity patterns enabled for recommendation, i.e., associated with contexts selected in the first step,
3) *Recommendation credibility* – validation of activity patterns in terms of recommendation credibility,
4) *Recommendation formulation* – formulation of the final recommendation and its justification following the formal model presented in Section 5.4.5,
5) *Recommendation ordering* – ordering of recommendation elements according to confidence indicator and social coefficient.

Enabled contexts

In the first step of the RMV method algorithm, a set of k contexts is identified referred to as an *enabled context* set. The smaller the context distance metric determined by the user-defined function $eval(co, co^{\alpha})$ (*cf.* Section 5.4.2), the more similar is the context to the context class.

For context class ${co_{\tau}}^{\alpha}$ and each context that supports at least one discovered activity pattern $z \in CMZ$ a context distance is calculated. Then a set of k activity pattern contexts with the lowest value of context distance is selected for further analysis.

Definition 5.59. (Recommendation Threshold). Let ω be a *recommendation threshold* stating the maximum accepted context distance between context and context class.

Let $contexts(CMZ, \ell)$ denote the set of all the unique contexts of all the activity patterns discovered from CMZ, i.e., $contexts\big(CMZ, \ell\big) = \{co(z) \mid co(z) \in fco(z), z \in CMZ\}$.

Definition 5.60. (Enabled contexts). *Enabled contexts* for event log ℓ is a set $enabled(\tau, co_\tau{}^\alpha) = \{co(z) \mid \mathrm{co(z)} \in \mathrm{contexts(CMZ,}\ \ell), eval(co(z), co_\tau{}^\alpha) \geq \omega\}$.

Let k be a user-defined size of target $enabled(\tau, co_\tau{}^\alpha)$ set such that $k<n$, where $n = \big|contexts(CMZ, \ell)\big|$.

The pseudo-code calculating the value of ω and finding elements of *enabled* set is presented in Listing 5.9.

Listing 5.9. Finding enabled context set

```
GetEnabledContexts (
    Set of activity patterns: CMZ,
    Number of searched neighbors: k;
    Context class: $co_\tau{}^\alpha$)
: enabled contexts set
{
    C ← contexts(CMZ, l);
    eC ← Ø;        /* contains a set of contexts; a maximum set size equals k ; */
    $\omega$ ← ∞;         /* recommendation threshold */

    for each c from C
    {
            val = $eval(co, co^\alpha)$;
            $\omega$ = get the smaller value of context distance from among contexts in eC set;
            if (val is smaller than $\omega$)
            {
                    eC ← c;                          /* add c to set eC */
                    if (size of eC set > k)          /* if a set eC is to big */
                    {
                            remove context with the highest distance to context class;

                    }
            }
    }/* end-for */
    return eC;
}
```

Enabled activity patterns

Definition 5.61. (Enabled activity patterns). *Enabled activity patterns* for event log ℓ is a set of activity patterns whose contexts are in $\text{enabled}(\tau, {co_\tau}^\alpha)$ set. Formally, $enabledAP(\tau, {co_\tau}^\alpha) = \{z \mid co(z) \in enabled(\tau, {co_\tau}^\alpha)\}$.

Let $enabled(\tau, {co_\tau}^\alpha, z)$ denote a set of contexts of activity pattern *z* in *enabled* set.

Let $enabledTraces(\tau, {co_\tau}^\alpha, z)$ denote a set of traces supporting activity pattern *z* in contexts from the *enabled* set.

Example 5.29. Consider an event log of 600 traces. There are 6 activity patterns discovered in the event log that appear in 30 different contexts. In a particular:

- activity pattern z_A is supported by 120 traces in 10 different contexts $co_{A1}, \dots, co_{A10}$,
- activity pattern z_B is supported by 75 traces in 8 contexts $co_{B1}, \dots, co_{B8}$,
- activity pattern z_C is supported by 70 traces in 5 contexts $co_{C1}, \dots, co_{C5}$,
- activity pattern z_D is supported by 60 traces in 5 context $co_{D1}, \dots, co_{D5}$,
- activity pattern z_E is supported by 70 traces in 5 contexts $co_{E1}, \dots, co_{E5}$ and
- activity pattern z_F is supported by 60 traces in 2 context co_{F1}, co_{F2}.

For simplicity, assume that each context is unique which in general is not always true. For *k* = *9*, calculation of context distance for each of 36 contexts leads to identification of the following set: $enabled(\tau, co_\tau^\alpha) = \{co_{A1}, co_{A2}, co_{A3}, co_{A4}, co_{B1}, co_{B2}, co_{B3}, co_{C1}, co_{D1}\}$. The context distance of every other context was greater than context distance of any context from the $enabled(\tau, {co_\tau}^\alpha)$ set. Thus, the set $nabledAP(\tau, {co_\tau}^\alpha) = \{z_A, z_B, z_C, z_D\}$.

The number of traces supporting each activity pattern *z* in context from $enabled(\tau, {co_\tau}^\alpha, z)$ is:

- for activity pattern z_A: 70 traces,
- for activity pattern z_B: 60 traces,
- for activity pattern z_C: 30 traces,
- for activity pattern z_D: 20 traces.

Thus, activity pattern z_A is considered to be the most frequent one in the enabled set. This situation is presented in Fig. 5.13, where each column corresponds to one activity pattern from $enabledAP(\tau, {co_\tau}^\alpha)$ set. Each column represents the number of traces supporting a particular activity pattern in a particular context. This information is supplemented by the distance of a particular context to the given context class. Numbers above columns represent the ratio of traces supporting an activity pattern in contexts from the $enabled(\tau, {co_\tau}^\alpha)$ set, to the total number of traces supporting a given activity pattern. For instance, activity pattern z_A is supported in context z_{A3} by 20 traces where the distance between z_{A3} and the context class is 3. Moreover, there are 50 traces that support z_A in other contexts than $z_{A1}, z_{A2}, z_{A3}, z_{A4}$. ▪

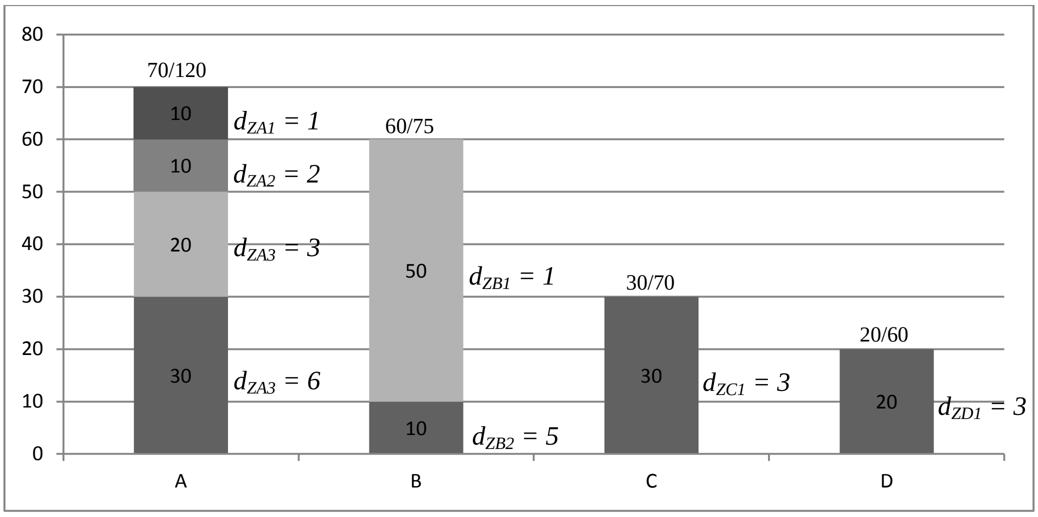


Fig. 5.13. Enabled activity patterns

Recommendation credibility

In this step, each enabled activity pattern from $enabledAP(\tau, co_\tau{}^\alpha)$ is validated for confidence of its recommendation. Validation of confidence is crucial to guarantee (1) robustness to noise data, (2) representative character of selected neighborhood, (3) limited influence of distant contexts on recommendation compared to similar contexts. In the RMV method, measures of *sensitivity*, *specificity* (Simon & Boring, 1990) and *weighted context distance* are used to validate the confidence of potential recommendation of each activity pattern from $enabledAP(\tau, co_\tau{}^\alpha)$ set.

Sensitivity specifies how many percent of event traces support the frequent activity pattern *z* in contexts from $enabled(\tau, co_\tau{}^\alpha)$ set. High sensitivity means that contexts from $enabled(\tau, co_\tau{}^\alpha)$ are typical for z.

Specificity is used to estimate cleanness of set $enabledAP(\tau, co_\tau{}^\alpha)$. The smaller $enabledAP(\tau, co_\tau{}^\alpha)$ set, the set of activity patterns suitable for a context class is more specific and recommendation is less fuzzy. Moreover, the more dominant the activity pattern *z* is in terms of support from among all the activity patterns in the $enabledAP(\tau, co_\tau{}^\alpha)$ set, the higher is *specificity*. Activity patterns of high *sensitivity* and *specificity* are preferred for recommendation.

Let $sensitivity(z, enabled(\tau, co_\tau{}^\alpha))$ denote a function returning *sensitivity* of activity pattern *z*.

Let $specificity(z, enabled(\tau, co_\tau{}^\alpha))$ denote a function returning *specificity* of activity pattern *z*.

Example 5.30. Continuing Example 5.29, *sensitivity* and *specificity* for z_A is:

$$sensitivity(z_A, enabled(\tau, co_\tau{}^\alpha)) = \frac{70}{120} = 58.3\%, \text{ and}$$

$$specificity(z_A, enabled(\tau, co_\tau{}^\alpha)) = \frac{600-(120+60+30+20)}{480} = 77\%.$$

If z_A was more dominant in *enabled* set, i.e, for example if only z_A and z_D were in the $enabledAP(\tau, {co_\tau}^\alpha)$ set, then:

$$sensitivity(z_A, enabled(\tau, {co_\tau}^\alpha)) = \frac{70}{120} = 58\%, \text{ and}$$

$$specificity(z_A, enabled(\tau, {co_\tau}^\alpha)) = \frac{600-(120+20)}{480} = 95.83\%.$$

Finally, if support of z_A for contexts from enabled set was lower and thus z_A was not so dominant in terms of frequency in the original $enabledAP(\tau, {co_\tau}^\alpha)$ set, i.e., if it was supported by 60 traces:

$$sensitivity(z_A, enabled(\tau, {co_\tau}^\alpha)) = \frac{60}{120} = 50\%, \text{ and}$$

$$specificity(z_A, enabled(\tau, {co_\tau}^\alpha)) = \frac{600-(120-60-30-20)}{490} = 75\%. ▪$$

Weighted context distance is used to minimize the influence of distant contexts on recommendation. Let $traces(z, co)$ denote the number of traces supporting activity pattern *z* in context *co*. To calculate *weighted context distance* of activity pattern *z*, for each *co* from $enabled(\tau, {co_\tau}^\alpha)$, the number of traces supporting activity pattern *z* in *co* is multiplied by a distance of *co* from the context class ${co_\tau}^\alpha$. Then, obtained products are added and divided by $|enabledTraces(\tau, {co_\tau}^\alpha, z)|$. The activity patterns with small weighted context distance are preferred for recommendation. Formally:

$$weightedContextDist(z, enabled(\tau, {co_\tau}^\alpha)) = \frac{\sum_{i=1}^{|enabled(\tau, {co_\tau}^\alpha, z)|} \mathrm{traces}(\mathrm{z}, \mathrm{co}_i) * \mathrm{eval}(co_i, {co_\tau}^\alpha)}{|enabledTraces(\tau, {co_\tau}^\alpha, z)|}$$

Let $weightedContextDist(z, enabled(\tau, {co_\tau}^\alpha))$ denote a function returning *weighted context distance* of activity pattern *z*.

Example 5.31. Calculate *weighted context distance* for z_A and z_B from Example 5.29:

$$weightedContextDist(z_A, enabled(\tau, {co_\tau}^\alpha)) = \frac{1*10+2*10+3*20+6*30}{70} = \frac{270}{70} = 3.18$$

$$weightedContextDist(z_B, enabled(\tau, {co_\tau}^\alpha)) = \frac{1*50+5*10}{60} = \frac{100}{60} = 1.66 ▪$$

Weighted context distance is definitely smaller for z_B. This means that the context class ${co_\tau}^\alpha$ is more typical for z_B than for z_A, i.e., z_B usually appears in contexts more similar to the context class. Still z_B does not appear in such contexts as often as z_A. It is left up to a user whether he/she prefers less frequent pattern appearing on average in more similar contexts or he/she prefers frequency over similarity.

Recommendation formulation

Let ς denote a user-defined minimum *sensitivity* of an activity pattern *z* from the set $enabledAP(\tau, {co_\tau}^\alpha)$ to be accepted for recommendation.

Definition 5.62. (Recommended Activity Patterns). A *recommended activity pattern* is a set:

$recomm(\tau, {co_\tau}^\alpha) =$
$\{z \in enabledAP(\tau, {co_\tau}^\alpha) | sensitivity(z, enabledTraces(\tau, {co_\tau}^\alpha, z)) > \varsigma\} \subseteq$
$enabledAP(\tau, {co_\tau}^\alpha)$.

For each activity pattern z from $recomm(\tau, {co_\tau}^\alpha)$, the following metrics are calculated:

- $cdif = calculateNRI(\tau, z) - calculateRI(\tau, z, enabledTraces(\tau, z, {co_\tau}^\alpha))$, where *calculateRI* is the function calculating *recom_index* for traces from $enabledTraces(\tau, z, {co_\tau}^\alpha)$ and *calculateNRI* is the function calculating *nonrecom_index* for traces from $enabledTraces(\tau, z, {co_\tau}^\alpha)$;
- $spec = specificity(z, enabled(\tau, {co_\tau}^\alpha))$ – *specificity*;
- $sens = sensitivity(z, enabled(\tau, {co_\tau}^\alpha))$ – *sensitivity*;
- $ws = weightedContextDist(z, recomm(\tau, {co_\tau}^\alpha))$ – *weighted context distance*;
- $s = sup(z)$ – value of z support in event log ℓ.

All these values are considered attributes of justification ℵ of recommendation element $\delta(\tau, z)$ (*cf.* Section 5.4.5).

Example 5.32. Assume that ς=50%. Only two activity patterns from Example 5.29 have sensitivity greater than ς, i.e., $sensitivity(z_A, enabled(\tau, {co_\tau}^\alpha)) = 58{,}3\%$ and $sensitivity(z_B, enabled(\tau, {co_\tau}^\alpha)) = 80\%$. Thus, $recomm(\tau, {co_\tau}^\alpha) = \{z_A, z_B\}$. Recommendation element for activity pattern z_A is:

$\delta(\tau, z_A) =$
$\langle z_A, recom_index, nonrecom_index, \langle just, = $
$\{\langle CDIF, cdif\rangle, \langle SPEC, 77\%\rangle, \langle SENS, 53{,}8\%\rangle, \langle WS, 3{,}18\rangle, \langle S, 20\%\rangle\}\rangle\rangle$. ▪

The pseudo-code finding the $enabledAP(\tau, {co_\tau}^\alpha)$ set and formulating recommendations is presented in Listing 5.10.

Listing 5.10. Finding enabled activity patterns and formulating recommendations

```
GetRecomm (
    Set of enabled context: enabled,
    Partial trace: $\tau$;
    Context class: $co_\tau{}^\alpha$;
    Expected sensitivity: $\varsigma$): set of recommendations
{
    enapbledAP ← Ø; /* contains a set of activity patterns */
    recomm← Ø; /* contains a set of recommendation */7.

    for each context co from enabled
    {
        z ← get activity patterns identified in context co
        enabledAP ← z;
        if (sensitivity(z, enapbledAP) is not smaller than $\varsigma$)
        {
            recomm ← z;
        }
    }/* end-for */
    for each activity pattern z from recomm
    {
        recom_index = $calculateRI\ (\tau, z)$;
        nonrecom_index = $calculateNRI\ (\tau, z)$;
        cdif = nonrecom_index – recom_index;
        spec = calculateSpecificity(z, $enabledTraces(\tau, co_\tau{}^\alpha, z)$, enabledAP );
        sens = calculateSensitivity(z, $enabledTraces(\tau, co_\tau{}^\alpha, z)$, enabledAP );
        ws =     calculateWeightedContextDistance(z, $enabledTraces(\tau, co_\tau{}^\alpha, z)$);
        s = sup(z);
        just = createJustification(cdif, spec, sens, ws, s);
        r = createRecommendationElement(z, recom_index, nonrecom_index, just);
        recomm ← r;
    }/* end-for */
    return recomm;
}
```

If $|recomm(\tau, co_\tau{}^\alpha)| > 1$, activity pattern from $recomm(\tau, co_\tau{}^\alpha)$ are merged into *generic activity pattern* which generalizes behavior observed in recommended activity patterns.

Let $gz\big(recomm(\tau, co_\tau{}^\alpha)\big)$ denote a *generic activity pattern* discovered from activity patterns from set *recomm*.

Let $L^\alpha_{se^\alpha}$ denote a set of relations adjacent to class of service entity se^α.

Let L_{se} denote a set of service relations adjacent to service entity se.

Definition 5.63. (Attribute equality of classes of service entities). Two classes of service entities se_1^α and se_2^α are *attribute equal* if they have the same set of attribute constraints.

Definition 5.64. (**Attribute equality of service entities).** Two service entities se_1 and se_2 are *attribute equal* if they have the same set of attributes.

Definition 5.65. (**Equality of service requirements).** Two service requirements l_1^α and l_2^α are equal if (1) they are adjacent to attribute equal classes of service entities, (2) they have the same set of attribute constraints.

Definition 5.66. (Equality of service relations). Two service relations l_1 and l_2 are equal if (1) they are adjacent to attribute equal service entities, (2) they have the same set of attributes.

Merging activity patterns from set $recomm(\tau, co_\tau{}^\alpha)$ is performed in the following manner:

1. *Merging process models* – information concerning temporal relationships among activities in each activity pattern $z \in recomm(\tau, co_\tau{}^\alpha)$ are used to discover a generalized process model capturing behavior of all the patterns from $recomm(\tau, co_\tau{}^\alpha)$; any existing algorithm discovering process model based on temporal relationships among activities can be used here, e.g. *Alpha* algorithm (Aalst, et al., 2004); each activity is associated with service description that together comprise service oriented description of a process model;
2. *Merging service network schemas* – service network schema of generic activity pattern is created by adding service requirements from activity patterns. Each requirement service in a generic activity pattern service network schema must be unique, i.e., if requirement services from different activity patterns are equal, requirement service is added only once. Unique service requirements are added together with adjacent classes of service entities. Equal classes of service entities associated with service requirements are merged, i.e. if two classes of service entities se_1^α and se_2^α are attribute equal, and the resulting class of service entity se_R^α is attribute equal to them, then set $L_{se_R}^\alpha = L_{se_1}^\alpha \cup L_{se_2}^\alpha$. Finally classes of service entities with no adjacent service requirements are also added to the set of service requirements in service network schema.
3. *Merging service network* – service network of generic activity pattern is created by adding service relations from activity patterns. All the equal service relations and service entity decryptions are merged. If two service entity description are attribute equal, then set $L_{se_R} = L_{se_1} \cup L_{se_2}$.
4. *Merging mappings* – merged classes of service entities are associated service descriptions assigned to corresponding classes in activity patterns. Service entity descriptions are assigned to corresponding classes of service descriptions from the activity patterns.

Example 5.33. Consider two activity patterns presented in Fig. 5.14 and Fig. 5.15 that were discovered and recommended for a particular context class. Each activity pattern encompasses definition of a process model, service oriented summary of a process model, service network schema and service network. Assume that each symbol in service network schema represents a unique attribute describing class of service entity or service requirement of an actor. The activity pattern following from the merge of these two patterns is presented in Fig. 5.16. In Fig. 5.16, the process model is represented as a Petri Net discovered from temporal relationships captured in activity patterns from Fig. 5.14 and Fig. 5.15. Petri Net is discovered using *Alpha* algorithm that aims at finding balance between four dimensions of process mining: generalization, simplicity, fitness, and precision. Although all the activities are captured in the model, it is not true for all the transitions among activities. Note that while according to Fig. 5.14, activities 2 and 4 can be executed in sequence, it is not possible in Fig.

5.15. Thus, according to Petri Net in Fig. 5.16 these two activities can be performed only in parallel. On the other hand, Petri Net in Fig. 5.16 permits infinite loop of two activities: activity 2 and some other unknown activity.

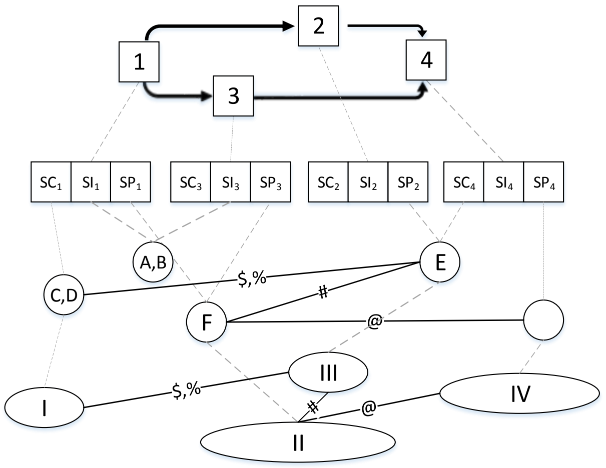


Fig. 5.14. Example of an activity pattern

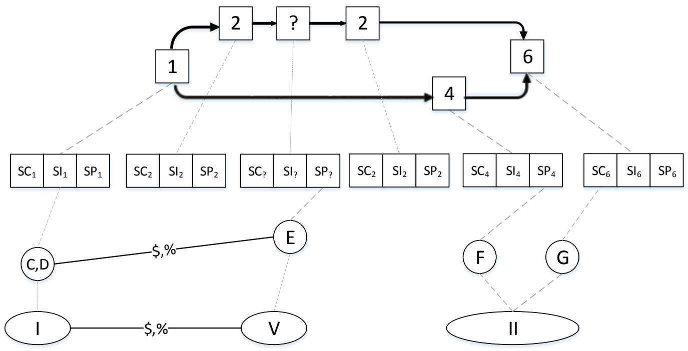


Fig. 5.15. Example of an activity pattern

While in activity patterns all the splits and joins follow AND condition, Petri Net adds also OR condition at splits and joints. For instance, once Activity 4 is executed, it is possible to continue either to Activity 6 or to the end. Service network schema and service network are also merged. Classes $\langle C, D\rangle$ are merged as they are the same in both activity patterns, i.e., the scope of attributes in classes of service entity, service requirements and their attributes are the same in both activity patterns. In both activity patterns, classes $\langle C, D\rangle$ are linked to two different service descriptions: sc_1 and si_1. In the merged generic activity pattern, the class is linked to both sc_1 and si_1. Note a different case of the $\langle F\rangle$ class. Although in both patterns such a class appears, the number and description of service requirements associated with it in each activity pattern is different. Thus, classes are not merged. The set of actors in service networks and corresponding social relations are copied from both activity patterns into one graph.

Generalization of behavior performed during activity pattern merge is also true for service network schema. For instance, while each activity pattern assigns $\langle C, D\rangle$ class to one service description, according to the generic activity pattern it is assigned to both, so an actor assigned to these classes can play two process roles instead of one.

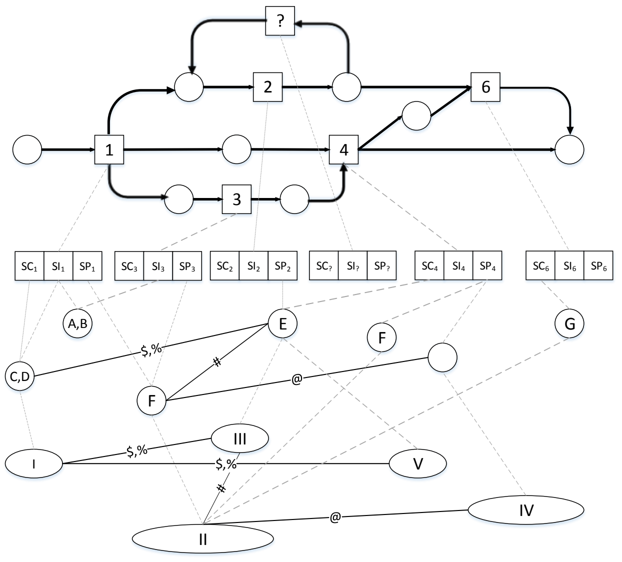


Fig. 5.16. Merged activity pattern with Petri Net as a process model

As mentioned in Section 3.1.3, service protocols and – as a consequence – activity patterns, abstract from a formal representation of a process model. For instance, in Fig. 5.17, generic activity pattern introduced in Example 5.33 has a process model represented by a process map including only frequent transitions among activities. ▪

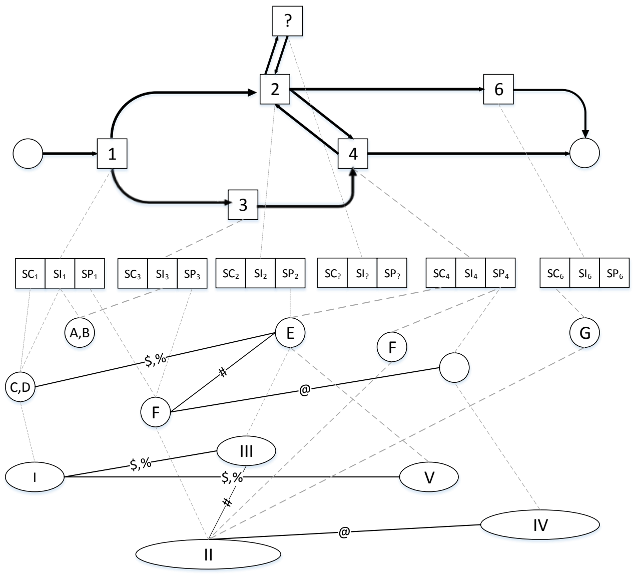


Fig. 5.17. Merged activity pattern with a frequent process map as a process model

Definition 5.67. (Full recommendation). The full recommendation is a tuple $\langle \text{gz}\left(recomm(\tau, {co_\tau}^{\alpha})\right), R(\tau) \rangle$.

Recommendation monitoring

In the basic scenario, it is assumed that recommendation elements generated by the RMV method are simply displayed to the user. In more advanced scenarios, recommended activity patterns are used for automatic orchestration of collaboration. In any case, the fact of following a recommendation by a set of collaborators is monitored by the Recommendation Monitor. The observations made by the Recommendation Monitor influence future recommendations. Fig. 5.18 is a simplified version of Fig. 5.2 – only those components of RMV methods associated with monitoring of recommendations and selection of activity patterns are presented.

The Recommendation Monitor associates recommended activity patterns with a group of collaborators that received recommendation in a particular process instance. The Recommendation Monitor observes events recorded in the event log (step 1 in Fig. 5.18) during further process instance execution. Periodically, a captured partial trace is validated in terms of supporting activity patterns (step 4). The validation is done using rules extracted from activity patterns. As an example, activity patterns can be parsed to one or more LTL rules (Aalst, et al., 2005). Definition of rule format and associated parser transforming activity pattern to rules is a part of the RMV method parameterization (*cf.* Section 5.6). Validation of each rule results in one of three statuses:

- *Rule satisfied* – execution of the process instance is in line with the rule;
- *Rule waiting* – the execution of activities described in the rule has not begun yet;
- *Rule violated* – execution of the process instance is not in line with the rule.

Example 5.34. Considered an activity pattern:

- *Trustex*, who is a *developer* with *10 year* experience and comes from *Poland*, uses service interface *Sign contract* categorized as *Init* with maximum duration of *7 days*, is used by to sign contract with a *company* that is usually a *general contractor* from *Poznań*; this takes place on *Monday*;
- Later, *Retrieve project plan* service interface categorized as *Init*, is used by a *general contractor* to interact with *architect Bud Tool* from *Berlin*.

Assuming, that the RMV method user uses *activity pattern-to-LTL* rule parser, the activity pattern is transformed to the following LTL rule:

```
formula activities_eventually_follow_each_other( ) :=

<> ( (

      ( SC == Trustex /\ SC-Type == Developer /\ SC-Country == Poland /\
      SC-Experience == 10 years /\ SI == Sign contract /\ SI-Category ==
      Init /\ SI-Duration == 7 days /\ SP-specialization == general
      contractor /\ SP-City == Poznań )

    /\ <>

      ( SC-Type == General contractor /\ SI == Retrieve project plan /\ SI-
      Category == Init /\ SP == Bud Tool /\ SP-City == Berlin /\ SP-
      specialization == Architect )

) );
```

The rule can be used by any of existing conformance checking methods based on LTL rules. ▪

The activity pattern is considered unsupported by a partial trace, i.e., activity pattern is not followed by a group of collaborators, if a particular rule derived from an activity pattern is not satisfied before the process instance completes, or before the next recommendation request for the process instance is issued. If the partial trace supports activity pattern, the priority of activity pattern for future recommendation increases (steps 6-10). If a pattern is not supported by a trace, its future priority decreases. Note that a partial trace can support more than one activity pattern from a recommendation. Thus, for one recommendation there can be a number of supported an unsupported activity patterns.

Recommendation elements ordering

To estimate the future value of an activity pattern in future recommendations, the Recommendation Monitor calculates two metrics, i.e., *confidence indicator* and *social coefficient*.

The range of *confidence indicator* is between 0 and 1. Confidence indicator of an activity pattern is increased every time a partial trace of a running VO collaborative process instance supports this activity pattern. Confidence indicator decreases if recommended activity pattern is not supported by the trace.

The value of *social coefficient* for a group of collaborators is based on classifier α. Calculation of social coefficient for a group of collaborators requires that each activity pattern from *recomm* is assigned to one of two categories: *group* or *general*. If majority of traces supporting a particular pattern *z* includes participation of at least half of collaborators involved in execution of a running process instance, then the pattern is assigned to the *group* category. Otherwise, it is classified as *general*. Typically, *general* activity patterns will come from process instances performed in other departments, units or organizations, but concerning the same subject.

The classifier α describes the collaborators group on a scale between 0 and 1, where 1 denotes a group always following activity patterns categorized as *group*. At the other end of the classifier ($\alpha = 0$), a group follows only *general* activity patterns. The value of α increases when the group carries out *group* recommendations. Similarly, the value of α is reduced when the group follows *general* recommendations. Reducing α reflects group's learning from other groups of collaborators. A group's work profile potentially deviates for one process instance to another. Once the Recommendation Monitor discovers that a particular pattern is supported by a partial trace of running process instance, a value of α for this group of collaborators and this running process instance is updated.

Note that while the confidence indicator is a feature of an activity patterns included in a recommendation, the social coefficient is a feature of a particular group of collaborators within a particular process instance that requests and accepts recommendations.

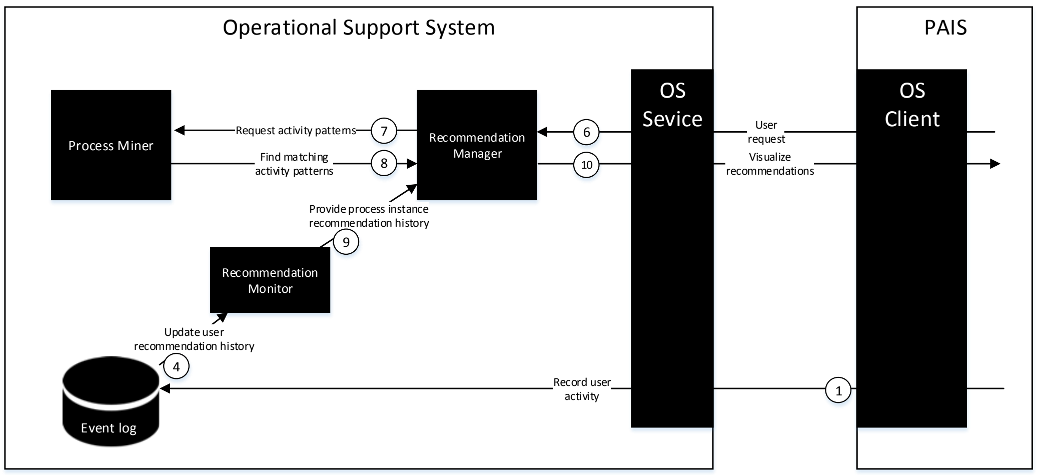

Operational Support System
PAIS
OS Sevice
OS Client
Process Miner
Request activity patterns
7
Recommendation Manager
6
User request
Find matching activity patterns
8
10
Visualize recommendations
Provide process instance recommendation history
9
Recommendation Monitor
Update user recommendation history
4
Event log
Record user activity
1

Fig. 5.18. Recommendations monitoring

Activity patterns returned by the Process Miner (step 7-8) are ordered by the Recommendation Manager before forwarding a recommendation to a group of collaborators (step 10). Ordering is performed according to the *recommendation value* calculated on the basis of *cdif* value, confidence indicator associated with recommendation element and social coefficient of each recommendation element. For *general* activity patterns, *recommendation value* V_δ is calculated as:

$$V_\delta = cdif * ci * (1 - \alpha)$$

For *group* activity patterns *recommendation value* V_δ is:

$$V_\delta = cdif * ci * \alpha$$

Example 5.35. Consider a set of recommendation elements in Tab. 5.11. For each activity pattern from each recommendation element, user-defined functions returned values presented in *recom_index* and *nonrecom_index* column. Assumed *confidence indicator* values are presented in column *ci*. Each pattern has assigned category *general* or *group*. A list of recommendations is presented to a group of collaborators with estimated classifier $\alpha = 0.6$ – preference towards *group* activity patterns. Thus, values in column *se* for *group* activity patterns equals 0.6 and for *general* activity patterns equal 0.4. ▪

Values in *cdif* are calculated by subtracting *recom_index* from *nonrecom_index*. They indicate the potential benefit of following a recommendation element. Negative value of *cdif* for activity pattern *E* indicates that although the activity pattern is performed frequently, it does not result in any benefit, i.e., the potential predicted cost of not following the activity pattern is smaller than when following it. Such activity pattern has negative *recommendation value* V_δ equal to -2,4. Information concerning such activity patterns is still useful – having knowledge of such "bad" pattern one may undertake actions to eliminate it. Patterns *A-D* have positive *cdif* value. The activity pattern with the highest estimated *recommendation values* is *C*. This activity pattern does not follow collaborators' group preferences, i.e., pattern is *general*, but it has high value of confidence indicator equal 1 and a relatively high potential benefit equal 20. The second highest estimation is for activity pattern B which is a *group* pattern but possible benefits and confidence are smaller than in the previous case. Activity pattern *D* has very high potential benefits but its confidence is low. Finally, activity pattern *A* has quite high confidence, but small benefits and it does not follow collaborators' group preference – it is a *general* pattern. The final ordering of patterns presented to a user is: 3, 2, 4, 1, 5. Increasing the social coefficient to $\alpha = 0.8$ would result in completely different order of recommendation elements: 2, 4, 3, 1, 5.

Tab. 5.11. Sorting of recommendation elements

Activity pattern id	***recom_ index***	***nonrecom_ index***	***cdif***	***ci***	***category***	***ce***	V_δ
A	55	60	5	0.8	general	0.4	0.8
B	35	50	15	0.7	group	0.6	6.3
C	20	40	20	1	general	0.4	8
D	10	60	50	0.2	group	0.6	6
E	30	10	-20	0.2	group	0.6	-2.4

Note that values of *confidence indicators* and *social coefficients* are not presented to a group of collaborators as a part of justification. Except *cdif* values, justification of each recommendation element includes only values of specificity (*spec*), sensitivity (*sens*), weighted context distance (*ws*) and support (*s*).

5.5.4. Activity Pattern Instantiation

In the RMV method, activity patterns are discovered on abstract, prototype or executable level (*cf.* Section 3.1.3). Both abstract and prototype activity patterns do not provide the complete mapping between service entities from the service network and classes of service entities in the service network schema. Such activity patterns must be instantiated and brought down to executable level. Both abstract and prototype activity patterns encompass definition of a set of partially ordered activities enriched with information concerning desired collaborators, their features and character of service relations among them. This information is a basis for assignment of service entities to classes of service entities.

In the RMV method, an instance of an activity pattern z is created by selection of service entities from the set of SOVOBE members. Selection is performed according to the constraints defined in service network schema σ_z^α. The selection of appropriate service entities is done from among all the service entities existing in SOVOBE, where selected entities must be instances of appropriate classes of service entities and must satisfy service requirements defined in the service network schema. Selection of service entities is performed on the basis of information about SOVOBE members provided via SOVOBE services including:

- Service entity descriptions encompassing information corresponding to service entity competences, capabilities, former experience, conspicuities, etc.
- Service relations among members of SOVOBE represented as service network σ_{SOVOBE}.

Note that σ_{SOVOBE} is not the same service network as the one discovered for activity pattern σ_z. Discovered σ_z is always a subnetwork of σ_{SOVOBE}. The aim of activity pattern instantiation is to add a subset of service entities from σ_{SOVOBE} to σ_z, so that potentially $\sigma_z \oplus \sigma_z^\alpha$ (*cf.* Definition A.11). Instantiation of an activity pattern is transformed to the following problem: given service network schema σ^α, find such subnetwork σ_z of social network σ_{SOVOBE} which maximizes the value of the $compliance(\sigma, \sigma^\alpha)$ function.

The RMV method algorithm for instantiation of activity patterns consists in four phases:

1. *Specification of VO* – definition or redefinition of classes of service requirements, service requirements and associated functions;
2. *Selection of service entities for classes of service entities* – selection of candidate service entities; the output of this phase is a set of service entities for each class of service entities from the activity pattern;
3. *VO variant generation* – ranking of service entity assignment according to the global compliance function; this phase includes generation and comparison of best possible VO variants, where a VO variant is a set of service entity descriptions assigned to the classes of service entity descriptions defined in phase 1 for execution of the activity pattern; as an output, a set of VO variants is generated ordered according to the values of the compliance function;
4. *VO inception* – registration of the VO in SOVOBE.

In each phase, interaction among selecting collaborators is assumed. In the first phase, the service network schema as well as the service network of an activity pattern may be redefined by selecting collaborators. Definition of VO specification encompasses:

- *Redefinition of classes of service entities and service requirements* – introduction of changes in the set of constraints associated with each class of service entity of a service requirement; also changes in predicates associated with constraints are possible or creation of new classes of service entities;
- *Reassignment of service entities to classes of service entities* – assignment of service entities can be changed, i.e., service entities can be removed or reassigned;
- *Class compliance function* – for each service class, definition of the function evaluating a compliance of a particular service entity to the class of service entity;
- *Class threshold values* – for each service class, an acceptable compliance level is assigned; this threshold is used in the second phase of MAAS to limit the number of assigned service entities;
- *Global compliance function* – global multi-attribute utility function is used in the third phase of the method to evaluate generated VO variants against satisfaction of service requirements;
- *Global threshold value* – an acceptable compliance level of global compliance function used to limit a number of VO variants passed to phase 4.

In the second phase, a set of services entities is selected for each class of service entity that has no assignment in the discovered activity pattern. Identified service entities are ordered according to the values of the class compliance function. Service entities with the value of compliance function smaller than the class compliance threshold are filtered out.

The number of potential VO variants that may be constituted with service entities identified in phase 2 is usually high. The goal of the third phase is to find an ordered list of VO variants ranked according to the compliance function in the usually large domain of potential VO variants. In phase 3, the genetic algorithm is used to determine the best fitted VO variants, as illustrated in Fig. 5.19. The genome *g* is an array of *N* items, *N* being the number of classes of service entities. Genes *re* are sets of service entity descriptions selected in phase 2 and assigned to classes of service entities *r* from the genome *g*.

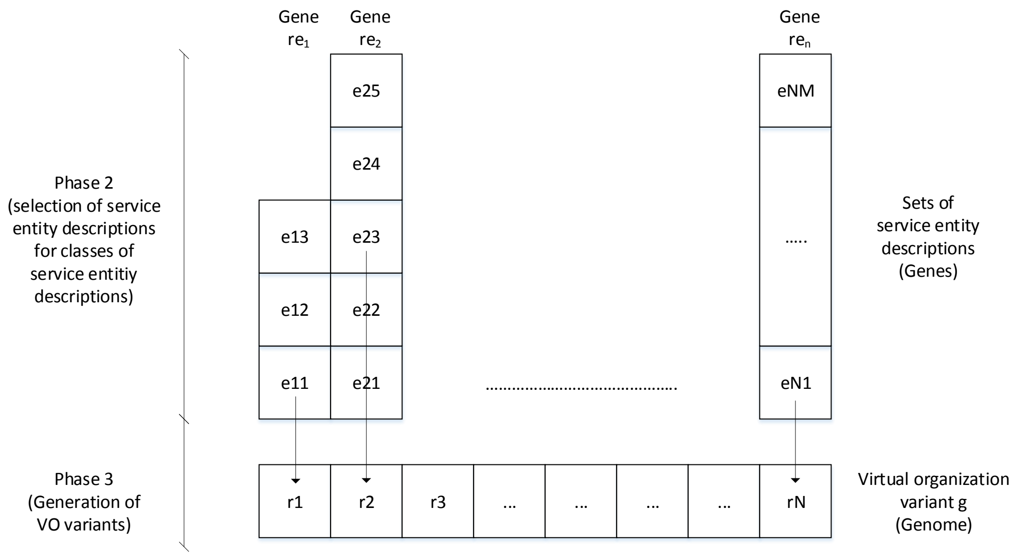

Gene re_1
Gene re_2
Gene re_n
Phase 2 (selection of service entity descriptions for classes of service entitiy descriptions)
e25
e24
e13
e23
e12
e22
e11
e21
eNM
.....
eN1
Sets of service entity descriptions (Genes)
Phase 3 (Generation of VO variants)
r1
r2
r3
...
rN
Virtual organization variant g (Genome)

Fig. 5.19. Phases 2-3 of activity pattern instantiation method

The crossover operator is the standard one-point crossover, while the mutation operator randomly selects a class of service entity description (a position in the genome) and randomly replaces the corresponding service entity description with another one among those available from the set of service entity descriptions assigned to this class in phase 2. In this way the genetic algorithm creates many VO variants. Each VO variant is evaluated by the use of the global fitness function defined in VO specification. The global compliance function used in this phase estimates the level of satisfaction of service requirements. As the result of this phase, a set of VO variants is selected ordered according to global compliance function values. A threshold value defined in phase 1 is used to filter out the VO variants: VO variants for which the value of the global compliance function is below the threshold value are not passed to phase 4.

In phase 4, the final selection of the one of the best ranked VO variants is made by collaborators. This VO variant is registered in the SOVOBE, i.e., service entity description and SOVOBE service network is updated.

5.6. RMV Method Parameterization

The RMV method includes a set of parameters required for its efficient operation. Some of these parameters, e.g., confidence indicator, or social coefficient, are deduced. Other parameters are defined by a user during RMV method configuration before its launch or they are submitted by a user to the RMV method in each recommendation request. User-defined parameters have impact on quality of recommendation and effectiveness of recommendation formulation. Definition of these parameters should take into account the characteristics of a specific VO collaborative process that the RMV method supports.

The summary of all the RMV method user-defined parameters is presented in Tab. 5.12. Parameters are grouped into three categories: event attributes, functions and constants.

Tab. 5.12. User-defined parameters of the RMV method

l.p.	Parameter name	Description	RMV method step
		Event attributes	
1.	Scope and types of activity instance description attributes	Given a log, where each event trace is described by *n* attributes, it is necessary to indicate which attributes must be taken into account by the RMV method. Next, selected attributes must be categorized according to event types presented in Section 5.4.3.	Activity sequence pattern discovery. Activity pattern identification
2.	Activity instance context scope	Definition of the scope of attributes describing activity process instance and event context. In particular, an attribute from the set of process instance attributes must be indicated as the one stating the outcome of the completed VO collaborative process instance.	Recommendation formulation
3.	Process instance attributes scope	Definition of the scope of attributes describing process instance.	Recommendation formulation
4.	Minimum activity instance	The minimum number of attributes that each activity description in an activity pattern must	Activity sequence pattern discovery

	description length	consists of.	
5.	Obligatory attributes	Names of attributes that must be included in each activity description from a discovered activity pattern.	Activity sequence pattern discovery
Functions			
1.	Context distance function *eval(co, co^{α})*	A function used to compare context class with a particular context. *Input*: context class, context.	Recommendation formulation
2.	Function *class(co)* mapping given context to context class	A function denoting context class *co^{α}* from context *co*. Note that *class(co)* function can simply convert context elements to context element constraints. In more advanced scenarios this function leads to generalization running process instance context *Input*: activity instance context.	Recommendation formulation
3.	*recom_index* and *nonrecom_index* calculation	Definition of two functions used for calculation of expected costs of following or not following recommendations. Calculation of *recom_index* and *nonrecom_index* values is based on cost attributes of collaborative events. It is also necessary to indicate which attribute associated with each event should be used during calculation. *Input*: set of process instances, cost attribute name.	Recommendation formulation
4.	Class compliance function	A function evaluating conformance of a service entity with a class of service entity. *Input*: class of service entity, service entity.	Activity pattern instantiation
5.	Global compliance function	A function evaluating conformance of a VO variant with a set of service requirements. *Input*: VO variant, set of service requirements.	Activity pattern instantiation
Constants			
1.	Class compliance threshold	A value used by the class compliance function to filter out service entities not suitable for assignment to a class of service entity.	Activity pattern instantiation
2.	Global compliance threshold	A value used by the global compliance function to filter out not matching VO variants.	Activity pattern instantiation
3.	Classification group size	The size of a classification enabled context set.	Recommendation formulation
4.	Minimum accepted sensitivity	Required value of sensitivity for the activity pattern to be accepted for recommendation.	Recommendation formulation
5.	Duration constraint	The maximum allowed time span between the start of the first and the completion of the last activity instance from an activity sequence supporting an activity pattern.	Activity sequence pattern discovery
6.	Gap constraint	The maximum time difference in milliseconds between two sequentially performed activities	Activity sequence pattern discovery

		from an activity sequence supporting an activity pattern.	
7.	Maximum timestamp constraint	The maximum completion time of the last activity from an activity sequence supporting an activity pattern.	Activity sequence pattern discovery
8.	Minimum sequence pattern length constraint	The minimum length of an activity pattern.	Activity sequence pattern discovery
9.	Maximum sequence pattern length constraint	The maximum length of an activity pattern.	Activity sequence pattern discovery
10.	Delta	Time interval used for identification of parallel events (in milliseconds)	Recommendation formulation
11.	Recommendation rule parser	The rule parser transforming activity patterns to rules which are used in recommendation monitoring	Recommendation monitoring
12.	Required activity pattern support level	The minimum support of an activity graph to be categorized as an activity pattern.	Activity sequence pattern discovery
13.	Time of validity	Time interval between discovery of activity patterns and update of activity pattern set in Process Miner	Activity sequence pattern discovery

A right set of parameters for both activity pattern discovery and recommendation formulation must be selected basing on knowledge concerning: (1) business domain, i.e., specificity of the process, (2) basic characteristics of the data set identified at the earlier phases of the method, e.g., number of discovered activity patterns influents parameters used for recommendation formulation such as context class function or classification group size.

5.7. RMV Method Computational Complexity

In this section, computational complexity of four parts composing the RMV method is presented: (1) activity sequence pattern discovery method, (2) activity pattern identification method, (3) recommendation formulation method, and (4) activity pattern instantiation method. Then, different ways aiming at reducing time of the RMV method execution are presented and explained. The following abbreviations are used to express the RMV method complexity:

- R – the number of traces in the event log,
- G – the number of events in the event log,
- N – the number of attributes in the event log,
- C – the number of attributes for each event,
- B – the average number of activity instance descriptions in each sequence,
- A – the number of identified sequence patterns,
- D – the number of activity pattern contexts,
- E – the number of service entities in the SOVOBE,
- l – the maximum number of events in one trace,
- w – the maximum activity instance description width,

- g – the number of genome generations in the genetic algorithm.

Complexity of activity sequence pattern discovery

Discovering activity sequence patterns is composed of three major steps: (1) discovery of activity sequences from the event log, (2) finding frequent 1-sequence patterns, and (3) finding closed sequence patterns.

Complexity of discovery of activity sequences in an event log is $O(G)$. The algorithm traverses all the events in all the traces, discovers activity instance descriptions and determines temporal relationships among them. In the worst case, the number of discovered activity instance description is equal to the number of events *G*.

The worst case of finding 1-sequence patterns in an even log arises, when each attribute in the event log is unique. Then, an event log can potentially generate up to 2^N-*1* frequent 1-sequence patterns, excluding the null set. The brute force approach to finding frequent 1-sequence patterns consists in determination of the support threshold (*cf.* Definition 5.35**)** for every candidate set of attributes. Such approach requires $G * (2^N - 1) * w$ comparisons. Thus, overall complexity is $O(2^N)$. To limit complexity of the brute force algorithm, the *Apriori* algorithm based on *a priori* principle is used in the RMV method (*cf.* Section 5.5.1). Complexity of the classic *Apriori* algorithm is $O(e^N)$, but recently it has been reduced to $O(N)$ (Suneetha & Krishnamoorti, 2010).

Complexity of finding closed sequence patterns in the RMV method is the same as that of the *PrefixSpan* algorithm. In the worst case, the number of frequent 1-sequence patterns is equal to the number of all the attributes in the event log, so complexity of this step is $O(N^l)$.

Overall complexity of activity sequence pattern discovery in the RMV method is determined by complexity of finding closed sequence patterns $O(N^l)$.

Complexity of activity pattern identification

In the RMV method, activity patterns are identified in all the discovered sequence patterns. In this section, first, complexity of identification of one activity pattern in one sequence pattern is presented. Then, overall complexity of activity pattern identification in the RMV method is presented.

Identification of an activity pattern in a sequence pattern consists of three steps: (1) identification of a service oriented summary of the process model, (2) identification of service network schema and corresponding mappings, (3) identification of the service network and corresponding mappings.

Complexity of identification of the service oriented summary of a process model is $O(B)$. The algorithm traverses all the activity instance descriptions in the sequence pattern and restores temporal relationships among them.

Identification of service network schema includes: (1) identification of classes of service entities, (2) identification of service requirements.

During identification of classes of service entities, the algorithm, first, traverses all the activity instance descriptions and discovers initial set of classes of service entities. Complexity of this step is $O(B)$. In the worst case, the number of classes of service entities is equal to *B*. Then, the algorithm compares all the initially discovered classes of service entities with each other looking for identical, redundant classes of service entities. The number of comparisons is $B * (B - 1)/2$. The number of comparisons of two classes of service entities depends on the number of attributes in each class. In the worst case, the number of attributes in each class of service entity is equal to C. Thus, the total number of comparisons of classes of service entities is $C * B * (B - 1)/2$. Overall complexity of identification of classes of service entities is $O(C * B^2)$.

In the worst case, if all the initially discovered classes of service entities are unique, the resulting number of classes of service entities is still equal to B. Mapping of classes to elements of service descriptions from the service oriented summary of a process model is $O(B)$. The algorithm visits each service description from the service oriented summary of the process model and maps elements of service description into classes of service entities.

During identification of service requirements, the algorithm also traverses all the activity instance descriptions and discovers the initial set of service requirements. Complexity of this step is $O(B)$. Then, the search for identical, redundant services requires $B * (B - 1)/2$ operations. The number of comparisons of two service requirements depends on the number of attributes indicated as service requirement descriptions, which in the worst case is equal to C. The total number of operations is then $C * B * (B - 1)/2$ and overall complexity is $O(C * B^2)$. Assignment of service requirements to classes of service entities requires validation of all the possible pairs of classes of service entities in terms of service relations existing among them. This needs $(B - 1)^2$ operations. Overall complexity of service network schema identification is $O(C * B^2)$.

Identification of service network includes: (1) identification of service entities, and (2) identification of service relations.

Calculation of the number of operations required to identify a service network is similar to identification of a service network schema. The search of duplicated service entities requires $B * (B - 1)/2$ comparisons. As each service entity is represented by its identifier, comparison of service entities does not require comparison of the sets of attributes. Complexity of mapping service entities to classes of service entities is $O(B)$. The algorithm visits all the classes of service entities and assigns corresponding service entities. Complexity of identification of service network is the same as complexity of identification of service requirements $O(C * B^2)$.

Complexity of activity pattern identification is determined by complexity of identification of service network schema $O(C * B^2)$. Overall complexity of activity pattern identification in A sequence patterns is $O(A * C * B^2)$.

Complexity of recommendation formulation

Complexity of finding the set of enabled contexts is $O(D)$. The algorithm traverses all the contexts and calculates the distance of each context from the context class.

Complexity of finding a set of enabled activity patterns for the set of enabled contexts is $O(A)$. The algorithm traverses all the discovered activity patterns and validates if at least one context from the enabled context set is the context of the activity pattern.

Ordering of activity patterns in a recommendation requires: (1) calculation of the recommendation value (*cf.* Section 5.5.3) for each enabled activity pattern, (2) sorting activity patterns according to the recommendation values. In the worst case, all the activity patterns are enabled. Complexity of calculation of recommendation values is $O(A)$. The algorithm calculates the recommendation value for each activity pattern from the enabled set. Complexity of activity pattern sorting using *QuickSort* algorithm is $O(A^2)$ in the worst case, and $O(A * log(A))$ on average, respectively.

Total complexity of recommendation formulation in the RMV method is determined by complexity of activity pattern sorting $O(A^2)$.

Complexity of activity pattern instantiation

Instantiation of an activity pattern is performed in two major steps: (1) selection of service entities for classes of service entities, (2) VO variant generation.

To find a candidate for classes of service entities, the algorithm takes each class of service entity, traverses all the service entities registered in the SOVOBE, and validates them in terms of compliance with the class of service entity. Complexity of this step is $O(B * E)$.

In the RMV method, the VO variant generation is performed using a genetic algorithm based on roulette wheel selection, point mutation, and one point crossover with both individuals and populations represented by fixed length tables. Complexity of this algorithm is $O(g * (mut + cross + select))$, where mut is point mutation complexity equal to $O(E * B)$, $cross$ is complexity of crossover equal to $O(E * B)$, and $select$ is complexity of selection equal to $O(E)$. Therefore, complexity of VO variants generation is $O(g * E * B)$.

Total complexity of activity pattern instantiation step in the RMV method is $O(g * E * C)$.

Total complexity

The summary of complexity of the RMV method steps is presented in Tab. 5.13. All the RMV method steps are performed sequentially. Total RMV method complexity depends on the highest complexity of its phases.

Tab. 5.13. Complexity of RMV method steps

RMV method parts	Complexity
Activity sequence pattern discovery	$O(N^l)$
Activity pattern identification	$O(A * C * B^2)$
Recommendation formulation	$O(A^2)$
Activity pattern instantiation	$O(g * E * B)$

Complexity assigned to activity sequence pattern discovery is higher than complexity of activity pattern identification, recommendation formulation and activity pattern instantiation. Thus, complexity of the RMV method is $O(N^l)$. This means that the number of event attributes in the event log, the length of traces and the time required to discover sequence patterns are key factors determining complexity of the RMV method.

Reduction of the RMV method execution time

The execution time of the RMV method is reduced by the use of parameters presented in Tab. 5.14. The presented set of parameters is a subset of parameters presented in Section 5.6.

Tab. 5.14. Parameters reducing the execution time of the RMV method

l.p.	Parameter name	Impact on execution time
1.	Scope of activity instance description attributes	Reduction of the set of activity instance description attributes reduces time of discovering 1-sequence patterns and the number of discovered 1-sequence patterns; smaller number of 1-sequence patterns reduces time of the closed sequence pattern discovery
2.	Minimum activity instance description length	Longer activity instance descriptions reduce the number of frequent 1-sequence patterns; smaller number of 1-sequence patterns reduces time of the closed sequence pattern discovery
3.	Obligatory attributes	High number of obligatory attributes reduces the number of frequent 1-sequence patterns; smaller number of 1-sequence patterns reduces time of the closed sequence pattern discovery
4.	Classification group size	Reduction of the classification group size reduces the size of the set of enabled contexts; as a consequence, the number of enabled activity patterns, the number of calculated recommendation values, and time needed to sort activity patterns are reduced
5.	Duration constraint	Shortening time span between activity pattern start and completion reduces the number of sequence patterns generated during sequence pattern discovery and the time of sequence pattern discovery
6.	Minimum accepted sensitivity	Increased value of acceptable sensitivity reduces the number of activity patterns used in a recommendation; as a consequence, the number of recommendation value calculations and time needed to sort activity patterns are reduced
7.	Gap constraint	Reduced maximum allowed time difference between activities reduces the number of sequence patterns generated during sequence pattern discovery; as a consequence, time of the closed sequence pattern discovery is reduced
8.	Maximum timestamp constraint	Reduced maximum completion time reduces the number of sequence patterns generated during sequence pattern discovery; as a consequence, time of the closed sequence pattern discovery is reduced
9.	Maximum sequence pattern length constraint	Reduced maximum allowed sequence pattern length reduces the number of sequence patterns generated during sequence pattern discovery; as a consequence, time of the closed sequence pattern discovery is reduced
10.	Required activity pattern support level	Increased support level decreases number of 1-sequences, decreases the length of discovered sequence patterns, and the time of sequence pattern discovery

6. Integration of the RMV Method with the *ErGo* System

In this section the prototype implementation of the RMV method is described and evaluated. First, the architecture of the prototype implementation of the RMV method is presented. Second, the RMV method is integrated with the PAIS named *ErGo*. Finally, the RMV method is evaluated on the real case data provided by company Epsilon.

6.1. RMV Method Prototype Architecture

The prototype implementation of the RMV method (RMV *prototype* in short) has been built to support collaborators by providing context-aware recommendations for their VO collaborative processes. The RMV method prototype: (1) discovers activity patterns in VO collaborative event logs and (2) recommends activity patterns suited for VO collaborative process context. The RMV prototype consists of *main* and *supporting* modules. The five *main* modules are the following: *Sequence Pattern Discovery (SPD)* module, *Activity Pattern Identification (API)* module, *Recommendation Formulation (RF)* module, *Activity Pattern Instantiation (APIN)* module, and *Recommendation Monitoring module (RM)*. Each module is the implementation of one of methods presented in Section 5.5. Functionality of two *supporting* modules is used by the main modules. The two supporting modules are: *Context (CON)* module and *Functions (FUNC)* module. The full list of modules is presented in Tab. 6.1. Each main module is mapped to logical component from Fig. 5.2.

Tab. 6.1. Mapping between modules and logical components

l.p.	Module	Logical component	Method
1.	SPD	Event Log Process Miner	Activity Sequence Pattern Discovery
2.	API	Process Miner	Activity Pattern Identification
3.	RF	Recommendation Manager	Recommendation Formulation
4.	APIN	MatchMaker	Activity Pattern Instantiation
5.	RM	Recommendation Monitor	Recommendation Formulation
6.	CON	–	Supporting module; classes modeling context
7.	FUNC	–	Supporting module; compliance functions

Interaction among the RMV prototype modules is presented in Fig. 6.1. The SPD module uses the CON module classes to capture contexts of activity instance descriptions and contexts of sequence patterns. The API module interacts with the SPD module to retrieve identified activity sequence patterns. The API module transforms activity sequence patterns into activity patterns. An activity pattern context is modeled in the CON module. The RF module is associated with the highest number of modules. The RF interacts with API to retrieve activity patterns that are the most suitable for recommendations. The RF module compares contexts of activity patterns using the CON module. Ordering of recommendation elements is based on user-defined functions from the FUNC module. The APIN module interacts with the API module to retrieve the activity pattern selected from the set of recommended activity patterns. All the functions used during activity pattern instantiation are modeled in the FUNC module. Finally, the RM module constantly exchanges information with the RF module concerning the use of recommendations. Classes of the RM module use classes coming from the SPD module in implementation of the monitoring mechanism.

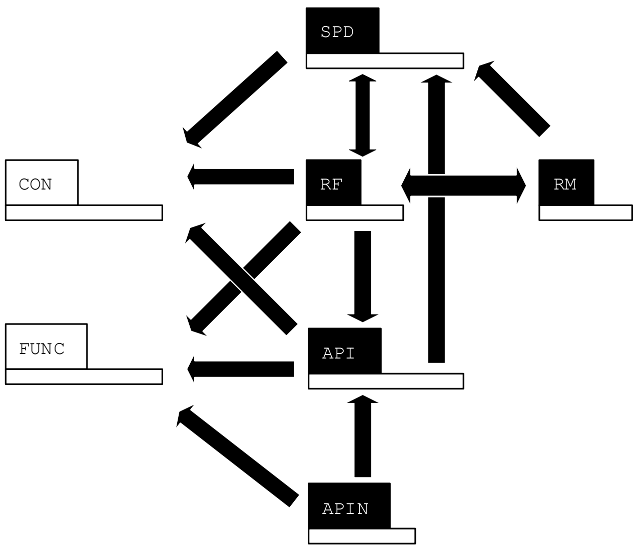


Fig. 6.1. Interactions between modules

The RMV prototype is distributed (Fig. 6.2). It is implemented as a client/server system: it is organized according to the user interface, business logic, and data storage tiers.

The business logic tier is implemented as the RMV *server*. The RMV server implements the logic of event log management, activity sequence pattern discovery, activity pattern identification, instantiation and recommendation, and recommendation monitoring. It is also responsible for access to the data storage tier. Functionality offered by the RMV server is externalized by OS Service (*cf.* Fig. 5.2). The Java^{TM} technology has been used for server implementation.

The data storage tier is implemented using the Oracle11i database.

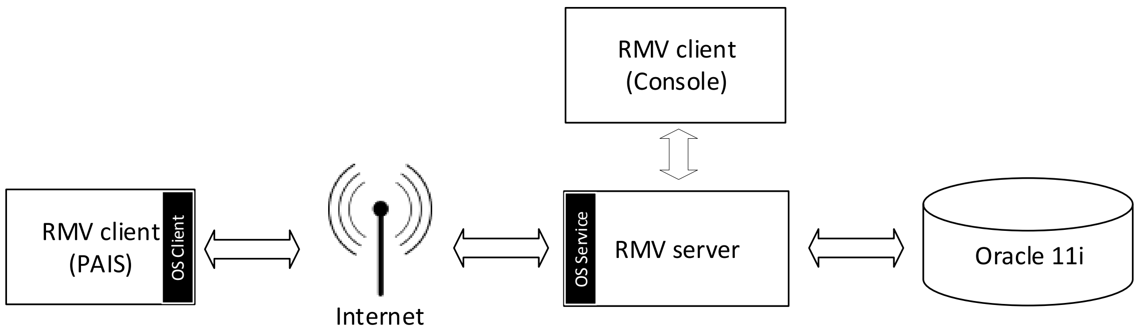


Fig. 6.2. Distribution of the RMV prototype

Functionality of the RMV prototype requires its integration with a PAIS system. The PAIS provides data concerning collaborators' activities and associated events. The *OS Client* (*cf.* Fig. 5.2) integrated with PAIS sends events via *OS Service* to the RMV server. Events are inserted to the database. PAIS events are used in activity pattern identification and monitoring of recommendation execution. PAIS provides also the presentation layer for the RMV prototype, i.e., PAIS presents recommendation outputs returned by the RMV prototype. Finally, it is assumed that PAIS has a repository of service entities that are used during activity pattern instantiation for search of the best matching collaborators and their services.

Communication between PAIS and the server follows the Web service standard. Other standards may be used as the RMV prototype is designed in a way independent of any specific middleware. For instance, integration with the *ErGo* system (*cf.* Section 6.2) is based on OSGi services.

Moreover, the RMV prototype comprises a *console client* that is a Java^{TM} application. The RMV console client provides a front-end to the APD, API, and RF modules. The RMV console functionality permits discovery of activity patterns in a given event log, and recommendation for a given partial trace. The RMV console functionality does not support activity pattern instantiation, recommendation monitoring, including management of confidence indicator and social coefficient values. Recommendation monitoring is possible only in case of integration with PAIS.

6.2. ErGo System Concept

The *ErGo* system (http://ergo.kti.ue.poznan.pl/) (Paszkiewicz, et al., 2011) (Paszkiewicz, et al., 2012) is an implementation of technical infrastructure supporting operation of SOVOBE and a proof of feasibility of the implementation of an IT system supporting the execution of VO collaborative processes based on discovery of activity patterns and their recommendations. The *ErGo* system has been developed in the Department of Information Technology at the Poznań University of Economics within the ITSOA project[12].

The *ErGo* system – from the Greek word èrgo ('ergo'), meaning "task", or "work" – aims at supporting collaboration between a real-estate developer and its subcontractors during the construction phase of a development process. The *ErGo* system takes into account the characteristics of the interactions between a real-estate developer and its subcontractors

[12] ITSOA project, https://www.soa.edu.pl/web/guest/home/

during the construction phase. A construction phase of a development process meets all the characteristics of VO collaborative process presented in Section 3.1.2.

In the *ErGo* system, VO collaborative processes are modeled as a set of activities performed by subcontractors on demand of a real-estate developer. Execution of an activity is regulated by contracts. If the realization of a development process instance meets an obstacle, the real-estate developer or a subcontractor may start adaptation, i.e., may request recommendations concerning selection of a set of activities, subcontractors and/or contracts related with the process.

Functionality of the *ErGo* system is provided by two categories of its components:

- *ErGo applications* – *ErGo* system components providing functionality available to end users through a graphical user interface. The list of applications include: *ErGo Organizations*, *ErGo Services*, *ErGo Investment Types* and *ErGo Investments*.
- *ErGo internal modules* – *ErGo* system components providing the core functions of the *ErGo* system such as: security, user management, KPI management, access to social network data, etc.; among others, the list of internal modules includes: *ErGo KPI*, *ErGo ServiceNet*.

The RMV method is tightly integrated with all the *ErGo* applications. The RMV method has been implemented partly as an *ErGo* internal modules and partly as an *ErGo* application:

- *ErGo Recomm* module – *ErGo* internal module being an implementation of the following RMV method elements: activity sequence pattern discovery, activity pattern identification, and recommendation formulation. This module is an implementation of *Process Miner*, *Recommendation Monitor*, *Recommendation Manager* and *OS Service* from Fig. 5.2; the *Recomm* module provides analysis of *ErGo* system event logs stored in the *ErGo Logs* aiming at discovery of activity patterns and provision of recommendations on request from other *ErGo* applications and modules;
- *ErGo Logs* module – *ErGo* internal module storing collaborative event logs; this module is an implementation of *Event log* from Fig. 5.2;
- *ErGo MatchMaker* application – implementation of an RMV activity pattern instantiation method; this application corresponds to *MatchMaker* component from Fig. 5.2.

The *ErGo* system architecture is presented in Fig. 6.3. The system architecture relies on the OSGi Container Equinox 3.5.2 deployed on the Jetty 6.1.19 application server. The infrastructural components of the system architecture are:

- OSGi 4.2 Blueprint Container providing means for management of OSGi services;
- Google Web Toolkit 2.0.3 with various libraries providing means for building graphical user interfaces;
- Apache CXF Distributed OSGi 1.2 providing means for building web services exposed by the system;
- Hibernate 3.3.1.GA providing means for communication with an Oracle 11g database.

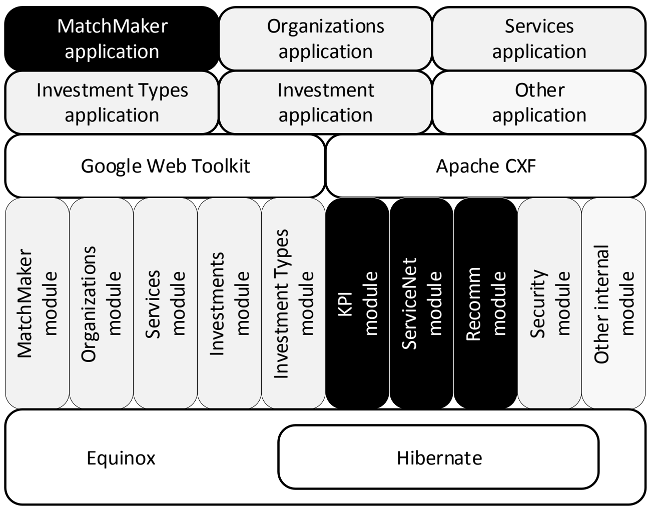


Fig. 6.3. System architecture diagram

Functionality provided by the *ErGo* system can be accessed by remote web services as well as through a graphical user interface. Two types of web services may be supported by Apache CXF Distributed OSGi:

- SOAP web services, relying on HTTP as a transport protocol and WSDL as a description language; currently supported in the *ErGo* system;
- RESTful web services, compliant with Java API for RESTful Web Services specification (JAX-RS).

The Web services issued by the *ErGo* system can be used by any external application supporting SOAP on HTTP. Detailed specification of the *ErGo* system is available on the project web site http://ergo.kti.ue.poznan.pl/documentation.

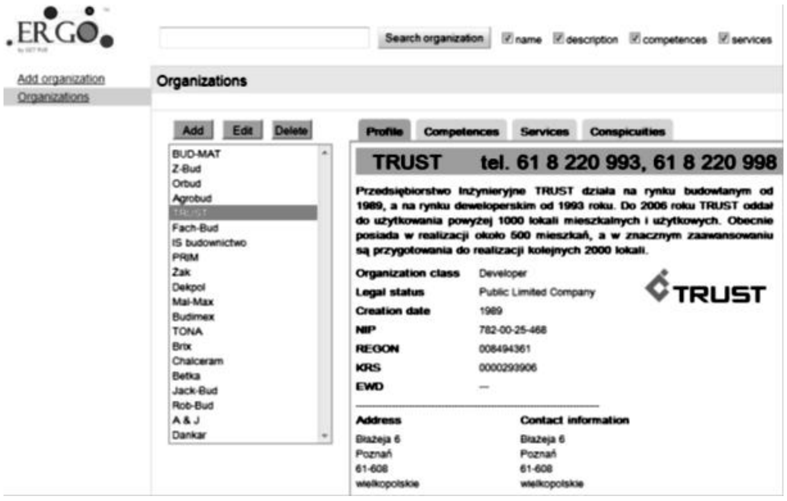
Below, in Section 6.3, *ErGo* applications together with two *ErGo* internal modules that are crucial for implementation and operation of RMV method modules, are presented in more detail. Then, in Section 6.4, integration of these applications with the RMV method components is presented.

6.3. ErGo Applications

The *ErGo* applications tackle various complementary aspects of the construction phase of development processes. The first application – *ErGo Organizations* – is responsible for management of the descriptions of organizations and their competences. The second application – *ErGo Services* – is responsible for management of the descriptions of business services provided by organizations. A real-estate developer manages all the activities and contracts with subcontractors using the third application – *ErGo Investments*. Templates for various types of development processes are defined in the fourth application – *ErGo Investment Types*. There are two internal modules important for operation of the RMV method components: *ErGo KPI*, and *ErGo ServiceNet*.

ErGo Organizations

The *ErGo Organizations* application permits users to manage the descriptions of organizations, especially their competences. A precise description of competences of organizations is important for management of development processes. Functionality of the *ErGo Organizations* application is organized as follows: first, a group of functions provide users with means for registering new organizations and updating the data concerning already registered organizations. Second, a group of functions permit users to manage the competences of organizations, based on the competence model proposed in (Paszkiewicz & Picard, 2011). Third, a group of functions give users means for retrieving organizations satisfying a set of requirements concerning either their profile or their competences. Finally, the *ErGo Organizations* application provides functions for searching and filtering registered organizations. A screen capture of the *ErGo Organizations* application is presented in Fig. 6.4. Users select an organization from the list of organizations on the left side. Detailed information about the selected organization is then displayed in the central panel. Competences of the organization are available on the associated tabbed panel. The *ErGo Organizations* application permits to find organizations meeting a particular set of requirements. This function is not available through the graphical user interface and it is not used by the user directly. Instead, it is offered in the form of services provided by the *ErGo Organization* module to be used by other *ErGo* system components, including RMV method components.



TRUST tel. 61 8 220 993, 61 8 220 998

Przedsiębiorstwo Inżynieryjne TRUST działa na rynku budowlanym od 1989, a na rynku deweloperskim od 1993 roku. Do 2006 roku TRUST oddał do użytkowania powyżej 1000 lokali mieszkalnych i użytkowych. Obecnie posiada w realizacji około 500 mieszkań, a w znacznym zaawansowaniu są przygotowania do realizacji kolejnych 2000 lokali.

Organization class	Developer
Legal status	Public Limited Company
Creation date	1989
NIP	782-00-25-468
REGON	008494361
KRS	0000293906
EWD	—

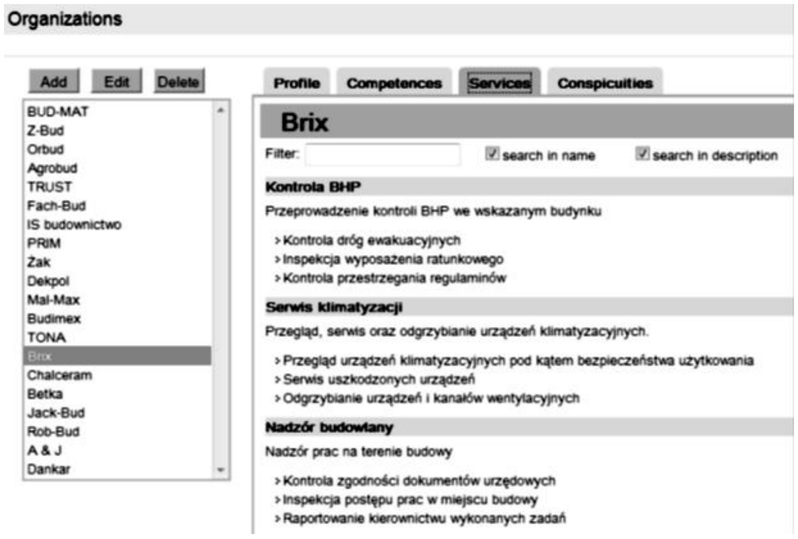
Address	Contact information
Błażeja 6	Błażeja 6
Poznań	Poznań
61-608	61-608
wielkopolskie	wielkopolskie

Fig. 6.4. *ErGo Organizations* application

ErGo Services

The *ErGo Services* application permits users to manage the descriptions of the services provided by organizations. The list of services that organizations provide is an important element of organization description. The choice of an organization as a subcontractor depends on services an organization provides. The goal of the *ErGo Services* application is to support management and search of services provided by organizations.

Functionality of the *ErGo Services* application is organized as follows: first, a group of functions provide users with means for registering new types of business services and updating the already registered ones. Second, a group of functions provide users with means for managing business services provided by organizations registered in the *ErGo Organizations* application. Third, a group of functions permit users to retrieve organizations satisfying a set of requirements concerning organizations' services. Finally, functions for searching and filtering business services registered in the ErGo *system* are provided by the *ErGo Services* application. Similarly to *ErGo Organizations*, functionality of finding services meeting a particular set of requirements is not available through the graphical user interface but offered as a service to other *ErGo* modules. A screen capture of the *ErGo Services* application is presented in Fig. 6.5. The graphical user interface of the *ErGo Services* application is a part of the *ErGo Organizations* application. For a given organization, details concerning the business services provided by it are available on the "Services" tabbed panel. The screen capture for management of service descriptions is presented in Fig. 6.6.



Filter:
[✓] search in name
[✓] search in description

Kontrola BHP

Przeprowadzenie kontroli BHP we wskazanym budynku

- Kontrola dróg ewakuacyjnych
- Inspekcja wyposażenia ratunkowego
- Kontrola przestrzegania regulaminów

Serwis klimatyzacji

Przegląd, serwis oraz odgrzybianie urządzeń klimatyzacyjnych.

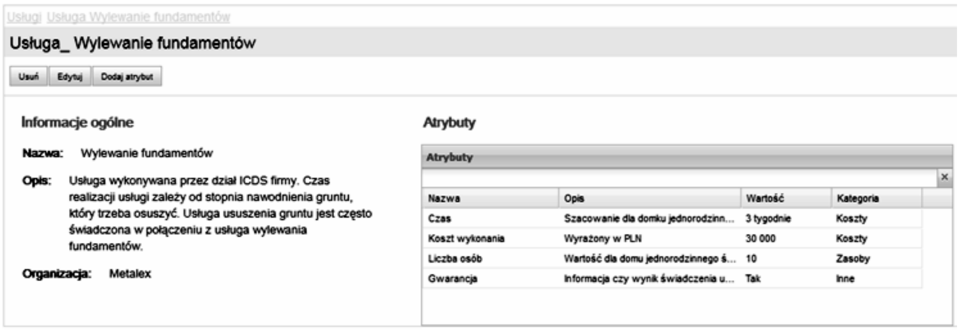
- Przegląd urządzeń klimatyzacyjnych pod kątem bezpieczeństwa użytkowania
- Serwis uszkodzonych urządzeń
- Odgrzybianie urządzeń i kanałów wentylacyjnych

Nadzór budowlany

Nadzór prac na terenie budowy

- Kontrola zgodności dokumentów urzędowych
- Inspekcja postępu prac w miejscu budowy
- Raportowanie kierownictwu wykonanych zadań

Fig. 6.5. *ErGo* Services main view



Usługa_ Wylewanie fundamentów

Informacje ogólne

Nazwa: Wylewanie fundamentów

Opis: Usługa wykonywana przez dział ICDS firmy. Czas realizacji usługi zależy od stopnia nawodnienia gruntu, który trzeba osuszyć. Usługa ususzenia gruntu jest często świadczona w połączeniu z usługa wylewania fundamentów.

Organizacja: Metalex

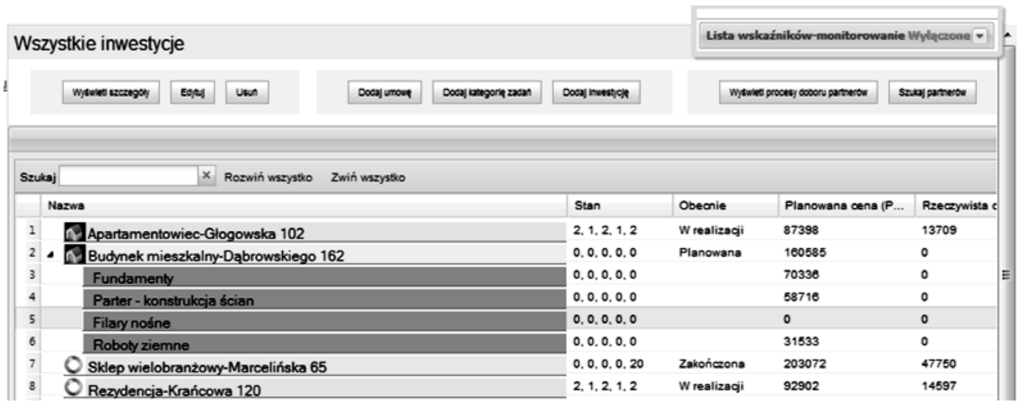
Atrybuty

Nazwa	Opis	Wartość	Kategoria
Czas	Szacowanie dla domku jednorodzinn...	3 tygodnie	Koszty
Koszt wykonania	Wyrażony w PLN	30 000	Koszty
Liczba osób	Wartość dla domu jednorodzinnego ś...	10	Zasoby
Gwarancja	Informacja czy wynik świadczenia u...	Tak	Inne

Fig. 6.6. *ErGo* Services – management of service description view

ErGo Investments

The *ErGo Investments* application permit users to manage activities and contracts for the construction phase of development processes. During the construction phase of a development process, the sequence of activities to be performed is usually not explicitly specified in advance. Moreover, a development process is usually not entirely determined when its construction phase starts. Activities to be performed during the construction phase of a development process are usually not planned ahead for more than three months. The *ErGo Investments* application permits users to control the realization of the construction phase of a development process in a seamless manner, by providing means for continuous definition of activities, preparation of contracts, appendices and progress payment claims. A screen capture of the *ErGo Investments* application is presented in Fig. 6.7.



	Nazwa	Stan	Obecnie	Planowana cena (P...	Rzeczywista c
1	Apartamentowiec-Głogowska 102	2, 1, 2, 1, 2	W realizacji	87398	13709
2	Budynek mieszkalny-Dąbrowskiego 162	0, 0, 0, 0, 0	Planowana	160585	0
3	Fundamenty	0, 0, 0, 0, 0		70336	0
4	Parter - konstrukcja ścian	0, 0, 0, 0, 0		58716	0
5	Filary nośne	0, 0, 0, 0, 0		0	0
6	Roboty ziemne	0, 0, 0, 0, 0		31533	0
7	Sklep wielobranżowy-Marcelińska 65	0, 0, 0, 0, 20	Zakończona	203072	47750
8	Rezydencja-Krańcowa 120	2, 1, 2, 1, 2	W realizacji	92902	14597

Fig. 6.7. *ErGo* Investments

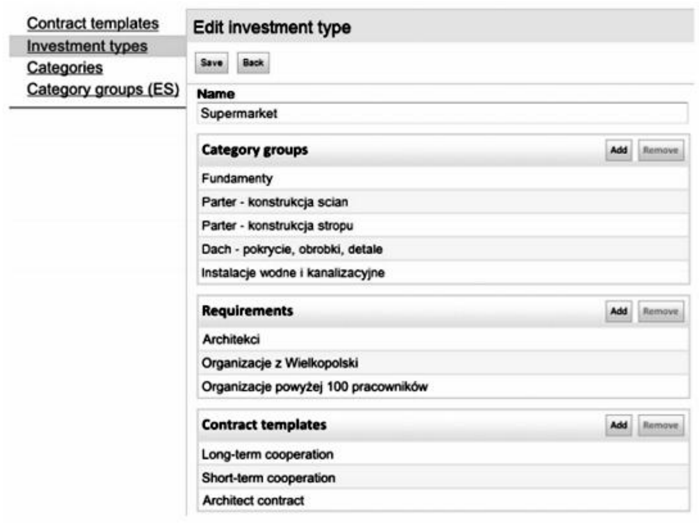
ErGo Investment Types

The *ErGo Investment Types* application permits users to manage reusable templates of process executions referred to as *investment types*. An investment type contains:

- *Activity descriptions* – descriptions of activities that may be included in a development process;
- *Category groups* – groups of descriptions of activities to be performed at the same stage of the construction phase;
- *Contract templates* – templates associated with activity descriptions that hold requirements concerning subcontractors and their services;
- *Contracts* – specify particular subcontractors and their services envisioned for execution of an activity; contracts are optionally assigned to contract templates.

Investment types are modeled as abstract or prototype service protocols (*cf.* Section 3.1.3). Each contract template is in fact a service description associated with a particular activity description. Requirements concerning subcontractors and subcontractors' services stored in a contract template correspond to classes of service entities associated with elements of service description. Contracts assign particular service entities, i.e., subcontractors and services, to their classes. Some investment types do not include contracts, i.e., service entities are not assigned to activity descriptions.

Investment types support planning the construction phase of a development process by reducing time required to plan development process execution, and promoting good practices from past executions of the development process. Functionality of the *ErGo Investment Types* application is organized as follows: first, a group of functions provide users with means for registering new investment types and managing already registered ones, including managing category groups, requirements, and contract templates. Second, a group of functions provide other *ErGo* modules and applications, e.g., *ErGo Investments*, with access to investment types. A screen capture of the *ErGo Investment Types* application is presented in Fig. 6.8. In the presented panel, an investment type concerning the construction of a supermarket building is presented. The investment type for supermarket contains five groups of activities, three requirements, and three contract templates that can be used to contract activities from subcontractors.


Contract templates
Investment types
Categories
Category groups (ES)

Edit investment type

Save Back

Name
Supermarket

Category groups
Fundamenty
Parter - konstrukcja scian
Parter - konstrukcja stropu
Dach - pokrycie, obrobki, detale
Instalacje wodne i kanalizacyjne

Requirements
Architekci
Organizacje z Wielkopolski
Organizacje powyżej 100 pracowników

Contract templates
Long-term cooperation
Short-term cooperation
Architect contract

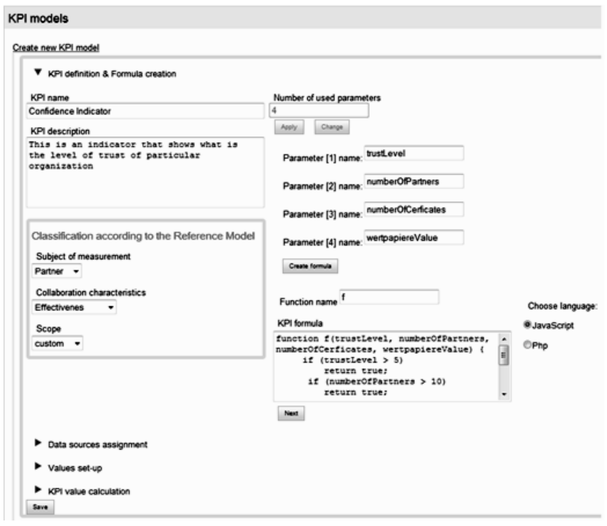
Fig. 6.8. *ErGo* Investments Types

ErGo Internal Modules

Two internal modules provide functionality necessary for the RMV modules: *ErGo ServiceNet* and *ErGo KPI*.

ErGo ServiceNet is an internal module that provides information about service relations among organizations and services registered in SOVOBE. Service relations are added to the module automatically as a result of actions performed by a user in other *ErGo* applications, or manually by an *ErGo* user. For instance, if an organization and its service are assigned to an activity, appropriate service relation is added to the module joining the subcontractor, the service and the real-estate developer. Service relations are typed and they contain a set of attributes. The *ErGo ServiceNet* module permits verification of relations among organizations. The module also validates the level of conformance of a set of organizations and services to a set of service requirements.

Module *ErGo KPI* supports definition of key performance indicators, where each indicator is a user-defined function. Function definition includes its logic, and the set of input and output parameters. A screen capture of the *ErGo KPI* is presented in Fig. 6.9.


KPI models

Create new KPI model

▼ KPI definition & Formula creation

KPI name: Confidence Indicator

KPI description:
```
This is an indicator that shows what is
the level of trust of particular
organization
```

Classification according to the Reference Model
Subject of measurement: Partner
Collaboration characteristics: Effectivenes
Scope: custom

Number of used parameters: 4

Parameter [1] name: trustLevel
Parameter [2] name: numberOfPartners
Parameter [3] name: numberOfCerficates
Parameter [4] name: wertpapiereValue

Function name: f

Choose language: [✓ JavaScript] Php

KPI formula:
```
function f(trustLevel, numberOfPartners,
numberOfCerficates, wertpapiereValue) {
     if (trustLevel > 5)
         return true;
      if (numberOfPartners > 10)
         return true;
```

▶ Data sources assignment
▶ Values set-up
▶ KPI value calculation

Fig. 6.9. Definition of KPI function in *ErGo KPI module*

6.4. Integration of the RMV Method with the ErGo system

The RMV method is implemented in the *ErGo* system as the *ErGo Logs* module, *ErGo Recomm* module and *ErGo MatchMaker* application. The RMV method components use functions provided by *ErGo Organizations*, *ErGo Services*, *ErGo Investments* applications, and *ErGo ServiceNet* and *ErGo KPI* modules.

ErGo Logs

ErGo Logs module inserts event related data into a database. In particular the following data are stored:

- Events associated with execution of an activity instance in the *ErGo Investments* application; and
- Attributes describing completed VO collaborative process instances.

Each event is stored in the database with the set of corresponding attributes in two separate tables: *event data* table, and *event context* table. For each event, the following information is stored in the *event data* table:

- Development process instance identifier, e.g., *bd24ff38-931d-4d69-8bcd-cr8ad24c3210,*
- Activity instance identifier, e.g., *bd24ff38-931d-4d69-8bcd-cr8ad24c3210,*
- Event identifier, e.g., *bd24ff38-931d-4d69-8bcd-cr8ad24c3210,*
- Event name, e.g., *Fundaments,*
- Transition, e.g., *Registered*,
- Subcontractor identifier, e.g., *ergo.organization.2761*,
- Service identifier, e.g., *ergo.service.3321*,
- Real-estate developer identifier, e.g., *ergo.organization.1*,
- Contract identifier, e.g., *ergo.contract.2762*,
- Contract template identifier, *e.g., ergo.contract.template.5678,*
- Timestamp: *Tue Dec 04 12:35:36 CET 2013*,
- Name of the *ErGo* internal module generating an event, e.g., *ergo.invetments.memory*,
- Identifier of the session of a user whose action was followed by event generation, e.g., *bd24ff38-931d-4d69-8bcd-cr8ad24c3210*.

Information concerning event context is stored in the *event context* table. This information captured for each event includes: development process instance identifier, event identifier, a set event context elements.

The set of process instance attributes that is assigned to the development process instance identifier is stored in the *process instance data* table.

ErGo Recomm

The *ErGo Recomm* module is responsible for analysis of event logs from the *ErGo Logs* module and their transformation to activity patterns. In the *ErGo* system, discovered activity patterns are investment types containing groups of activities, requirements and contract templates.

Configuration of the *ErGo Recomm* module includes all the parameters described in Section 5.6. In addition, configuration of the *ErGo Recomm* module encompasses:

- Indication of attributes from the *ErGo Organizations*, *ErGo Services*, *ErGo ServiceNet*, *ErGo Investment Types*, and *ErGo Investments* components to be incorporated into event traces;
- Indication of attributes from the *ErGo Investments* application to be incorporated into event context elements and process instance attributes;
- Indication of functions defined previously in the *ErGo KPI* to be used in various steps of the RMV method, e.g., indication of functions calculating *recom_index* and *nonrecom_index*.

Note that the *ErGo Logs* module does not hold all the attributes of subcontractors, services, contract templates, and contracts associated with a particular executed activity instance. Instead, it stores their identifiers in corresponding *ErGo* applications. Before data from the *ErGo Logs* module is used by *ErGo Recomm* module in activity pattern discovery, it is transformed to collaborative process log. Each trace from the event log is supplemented with data coming from other *ErGo* modules, i.e., attributes holding identifiers of service entities from *ErGo Organizations*, *ErGo Services* and *ErGo* S*erviceNet* are supplemented with a set of attributes actually describing service entities, and identifiers of contracts, while contract templates are supplemented with data from *ErGo Investments* and *ErGo Investment Types*.

Example 6.1. An example of simplified data from the *ErGo* collaborative event log created in the *ErGo Recomm* module is presented in Tab. 6.2 and Tab. 6.3. A set of attributes describing each process instance is presented in Tab. 6.4.

In Tab. 6.2, attributes that appear in the log are: process instance identifier (*pi-id*), activity instance identifier (*ai-id*), event identifier (*e-id*), event timestamp (*time.*) and event transition (*trans.*). Service customer, service interface and service provider unique identifiers are represented by columns *SC*, *SI* and *SP*, respectively. In the *ErGo* system all the activities are executed on demand of a real-estate developer. Thus, the value of *SC* attribute is the same for all the events. Characteristics of service entities are provided as values of attributes *SC-Type*, *SI-Duration*, and *SP-Specialization*. Two columns correspond to social relations existing among service providers – *SP-Country* and *SP-City*, e.g., all the companies, except *Bud Tool* share the relation of being located in *Poland*. Similarly, services are grouped into categories (*SI-Category* attribute). For the clarity, column corresponding to *ErGo* module name and user session are not presented in Tab. 6.2.

Event context elements recorded for each event presented in Tab. 6.3 include the day of week and season of year. A set of attributes describing each process instance is presented in Tab. 6.4.

Tab. 6.2. RMV event data table

pi-id	ai-id	e-id	Time	Trans.	SC	SI	SP	SC-Type	SP-Specialization	SP-Country	SP-City	SI-Category	SI-Duration
1	1	09	1	Start	Trustex	Fundamenty	City hall	Developer	Ground works	Poland	Poznań	Init	30 days
1	1	10	2	Complete	Trustex	Fundamenty	Arch Dome	Developer	Ground works	Poland	Poznań	Init	7 days
1	2	11	3	Start	Trustex	Parter – konstrukcja ścian	Bud Tool	Developer	Electrical installations	Germany	Berlin	Init	10 days
1	2	12	4	Complete	Trustex	Parter – konstrukcja stropu	Bud Tool	Developer	Electrical installations	Poland	Warsaw	Init	10 days
2	3	13	2	Complete	Trustex	Dach	Skan Bud	Developer	Vertical surfaces	Poland	Poznań	Init	7 days
2	4	14	3	Start	Trustex	Dach	Bud Tool	Developer	Electrical installations	Germany	Berlin	Init	1 day
3	5	15	1	Start	Trustex	Instalacje wodne i kanalizacyjne	Arch Dome	Developer	Ground works	Poland	Cracow	Final	6 days

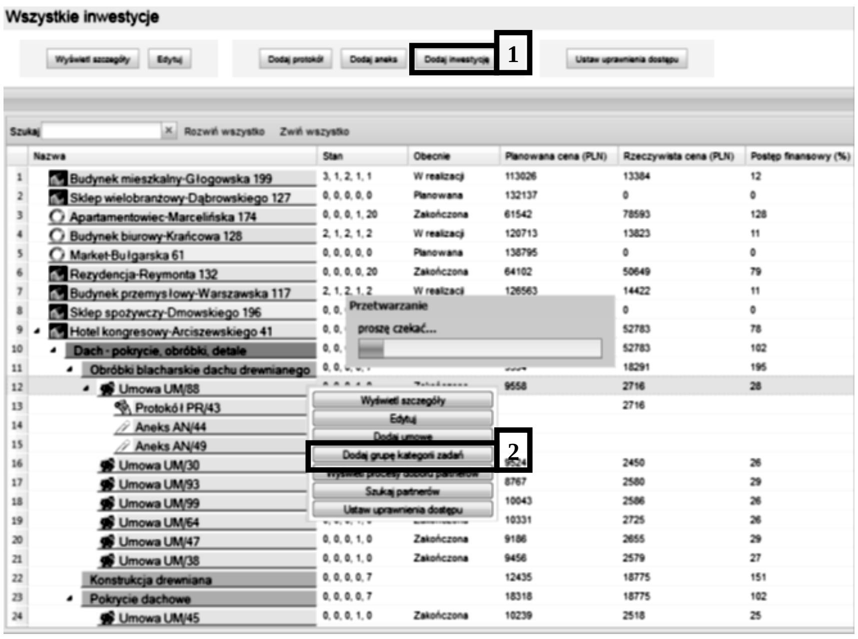
Tab. 6.3. RMV event context table

pi-id	e-id	Day	Season
1	09	Friday	Spring
1	10	Monday	Spring
1	11	Friday	Spring
1	12	Tuesday	Spring
2	13	Friday	Spring
2	14	Tuesday	Summer
3	15	Monday	Summer

Tab. 6.4. RMV process instance attributes

pi-id	Duration	Start	Complete	No. subcontractors	In time
1	9	1	10	10	true
2	3	1	4	12	true
3	5	4	9	5	false

The graphical user interface components presenting recommendation results are integrated into *ErGo Investment Types* and *ErGo Investments* applications. When a user attempts to add a new investment or a new category group to existing investments, a recommendation is provided to the user automatically. Creation of a new investment or a category group requires selection of an investment type. The selected investment type is used as a template for a new investment or a category group. The screen capture in Fig. 6.10 presents the *ErGo Investments* functionality of adding a new investment (1) and a new category group (2).

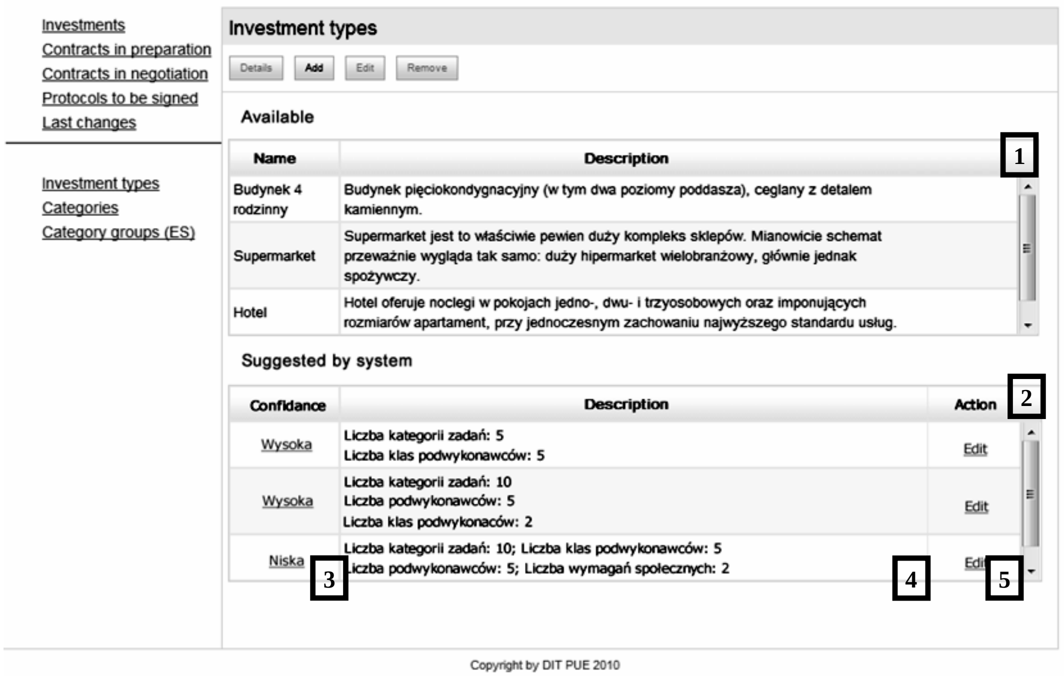


Wszystkie inwestycje

	Nazwa	Stan	Obecnie	Planowana cena (PLN)	Rzeczywista cena (PLN)	Postęp finansowy (%)
1	Budynek mieszkalny-Głogowska 199	3, 1, 2, 1, 1	W realizacji	113026	13384	12
2	Sklep wielobranżowy-Dąbrowskiego 127	0, 0, 0, 0, 0	Planowana	132137	0	0
3	Apartamentowiec-Marcelińska 174	0, 0, 0, 1, 20	Zakończona	61542	78593	128
4	Budynek biurowy-Krańcowa 128	2, 1, 2, 1, 2	W realizacji	120713	13823	11
5	Market-Bułgarska 61	0, 0, 0, 0, 0	Planowana	138795	0	0
6	Rezydencja-Reymonta 132	0, 0, 0, 0, 20	Zakończona	64102	50649	79
7	Budynek przemysłowy-Warszawska 117	2, 1, 2, 1, 2	W realizacji	126563	14422	11
8	Sklep spożywczy-Dmowskiego 196	0, 0, [illegible]	[illegible]	[illegible]	0	0
9	Hotel kongresowy-Arciszewskiego 41	0, 0, [illegible]	[illegible]	[illegible]	52783	78
10	Dach - pokrycie, obróbki, detale	0, 0, [illegible]	[illegible]	[illegible]	52783	102
11	Obróbki blacharskie dachu drewnianego	0, 0, [illegible]	[illegible]	[illegible]	18291	195
12	Umowa UM/88	[illegible]	[illegible]	9558	2716	28
13	Protokół PR/43	[illegible]	[illegible]		2716	
14	Aneks AN/44	[illegible]	[illegible]			
15	Aneks AN/49	[illegible]	[illegible]			
16	Umowa UM/30	[illegible]	[illegible]	9524	2450	26
17	Umowa UM/93	[illegible]	[illegible]	8767	2580	29
18	Umowa UM/99	[illegible]	[illegible]	10043	2586	26
19	Umowa UM/64	[illegible]	[illegible]	10331	2725	26
20	Umowa UM/47	0, 0, 0, 1, 0	Zakończona	9186	2655	29
21	Umowa UM/38	0, 0, 0, 1, 0	Zakończona	9456	2579	27
22	Konstrukcja drewniana	0, 0, 0, 0, 7		12435	18775	151
23	Pokrycie dachowe	0, 0, 0, 0, 7		18318	18775	102
24	Umowa UM/45	0, 0, 0, 1, 0	Zakończona	10239	2518	25

Fig. 6.10. Adding a new category group and a new investment in the *ErGo Investment* module

An investment type is selected from the set of investments types available in the *ErGo* system. Optionally, a user may define a new investment type. The screen allowing a user to select an existing investment type is presented in Fig. 6.11. Two sets of investment types are presented. In the “Available” table (1), there are investment types previously added to the system by a user. Those investment types are described in a meaningful way. In “Suggested by system” table (2), there are investment types discovered and recommended by the *ErGo Recomm* module. Each investment type is described by level of recommendation confidence of provided recommendation (3) and basic information concerning the number of categories and subcontractors in the recommended investment type (4). Finally, a user may select an investment type and display detailed information form the third column (5).



Investment types

Available

Name	Description
Budynek 4 rodzinny	Budynek pięciokondygnacyjny (w tym dwa poziomy poddasza), ceglany z detalem kamiennym.
Supermarket	Supermarket jest to właściwie pewien duży kompleks sklepów. Mianowicie schemat przeważnie wygląda tak samo: duży hipermarket wielobranżowy, głównie jednak spożywczy.
Hotel	Hotel oferuje noclegi w pokojach jedno-, dwu- i trzyosobowych oraz imponujących rozmiarów apartament, przy jednoczesnym zachowaniu najwyższego standardu usług.

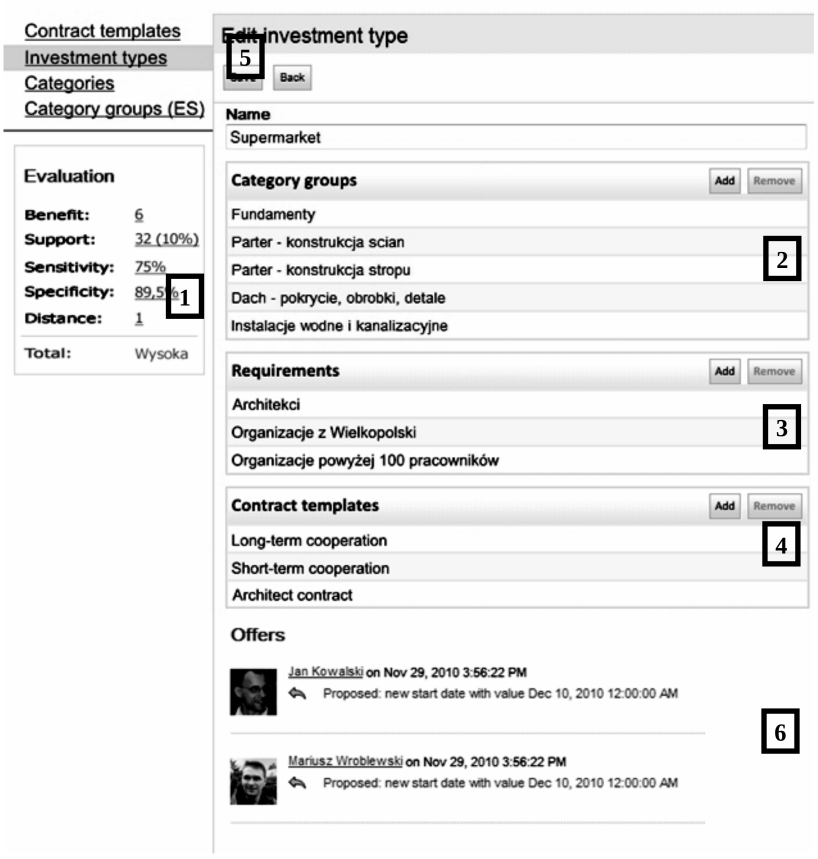
Suggested by system

Confidance	Description	Action
Wysoka	Liczba kategorii zadań: 5 Liczba klas podwykonawców: 5	Edit
Wysoka	Liczba kategorii zadań: 10 Liczba podwykonawców: 5 Liczba klas podwykonaców: 2	Edit
Niska	Liczba kategorii zadań: 10; Liczba klas podwykonawców: 5 Liczba podwykonawców: 5; Liczba wymagań społecznych: 2	Edit

Copyright by DIT PUE 2010

Fig. 6.11. A view of available and recommended investment types

In Fig. 6.12, a screen for viewing details of recommended investment type is presented. Details of recommendation are visible in the panel on the left (1). The screen provides possibilities to modify an investment type composed of activities (2), requirements and contract templates (4), and to save it (5) in the *ErGo* system. Finally, selection of the investment type is done in a collaborative manner. The discussion mechanism (6) built-in the *ErGo* system supports the exchange of comments and suggestion of possible modifications concerning the investment type.


Contract templates
Investment types
Categories
Category groups (ES)

Evaluation

Benefit:	6
Support:	32 (10%)
Sensitivity:	75%
Specificity:	89,5%
Distance:	1
Total:	Wysoka

Edit investment type

Name: Supermarket

Category groups
- Fundamenty
- Parter - konstrukcja scian
- Parter - konstrukcja stropu
- Dach - pokrycie, obrobki, detale
- Instalacje wodne i kanalizacyjne

Requirements
- Architekci
- Organizacje z Wielkopolski
- Organizacje powyżej 100 pracowników

Contract templates
- Long-term cooperation
- Short-term cooperation
- Architect contract

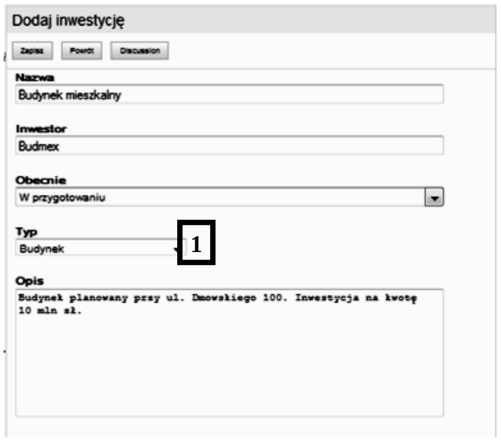
Offers

Jan Kowalski on Nov 29, 2010 3:56:22 PM
Proposed: new start date with value Dec 10, 2010 12:00:00 AM

Mariusz Wroblewski on Nov 29, 2010 3:56:22 PM
Proposed: new start date with value Dec 10, 2010 12:00:00 AM

Fig. 6.12. Adding recommended investment type to the *ErGo* system

In Fig. 6.13, a screen for creating an investment based on investment type (1) is presented. Created investment is added to the list of investments visible in Fig. 6.10. Similarly, a created category group is added to the list of category groups of a particular ongoing investment process.


Dodaj inwestycję

Nazwa: Budynek mieszkalny

Inwestor: Budmex

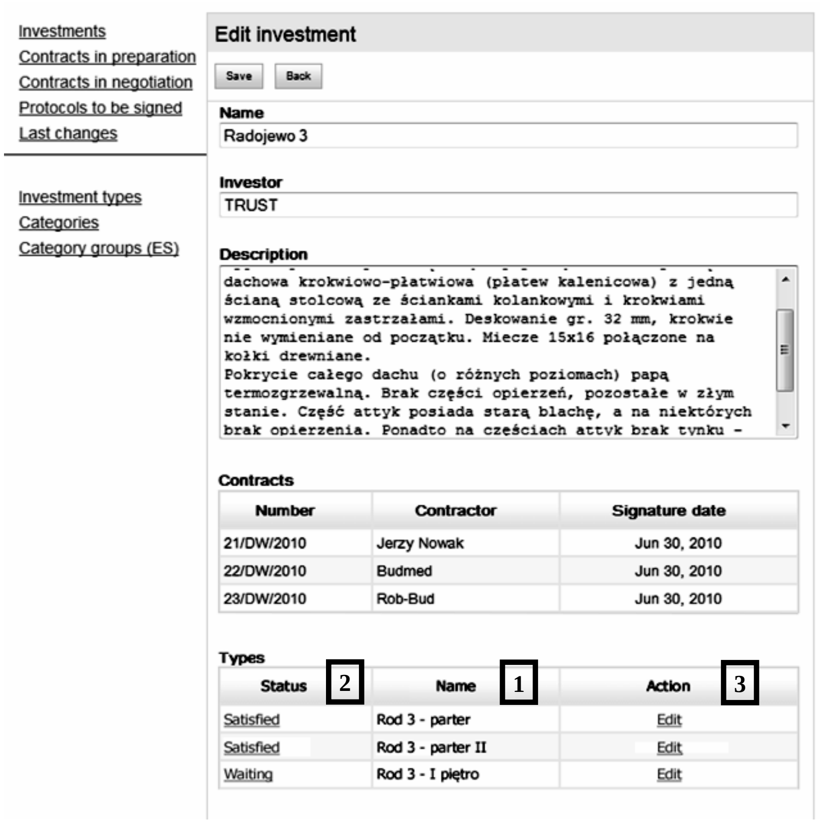
Obecnie: W przygotowaniu

Typ: Budynek

Opis: Budynek planowany przy ul. Dmowskiego 100. Inwestycja na kwotę 10 mln zł.

Fig. 6.13. Adding investment based on investment types

After the assignment of the investment type to the investment, the investment is monitored to validate if its execution is conducted in line with the investment type specification. In Fig. 6.14, the investment details view is presented. Among others, the view provides information concerning investment types assigned to the investments. Each investment type is described with its name (1) and status of its execution (2). Status *Satisfied* in Fig. 6.14 means that all the activities from the investment type were already executed in line with all the requirements specified in the investment type. *Waiting* status means that execution of activities from the investment type has not begun yet. Additional *Action* column (3) permits edition of detailed information about the investment type.



Name
Radojewo 3

Investor
TRUST

Description
dachowa krokwiowo-płatwiowa (płatew kalenicowa) z jedną ścianą stolcową ze ściankami kolankowymi i krokwiami wzmocnionymi zastrzałami. Deskowanie gr. 32 mm, krokwie nie wymieniane od początku. Miecze 15x16 połączone na kołki drewniane.
Pokrycie całego dachu (o różnych poziomach) papą termozgrzewalną. Brak części opierzeń, pozostałe w złym stanie. Część attyk posiada starą blachę, a na niektórych brak opierzenia. Ponadto na częściach attyk brak tynku -

Contracts

Number	Contractor	Signature date
21/DW/2010	Jerzy Nowak	Jun 30, 2010
22/DW/2010	Budmed	Jun 30, 2010
23/DW/2010	Rob-Bud	Jun 30, 2010

Types

Status	Name	Action
Satisfied	Rod 3 - parter	Edit
Satisfied	Rod 3 - parter II	Edit
Waiting	Rod 3 - I piętro	Edit

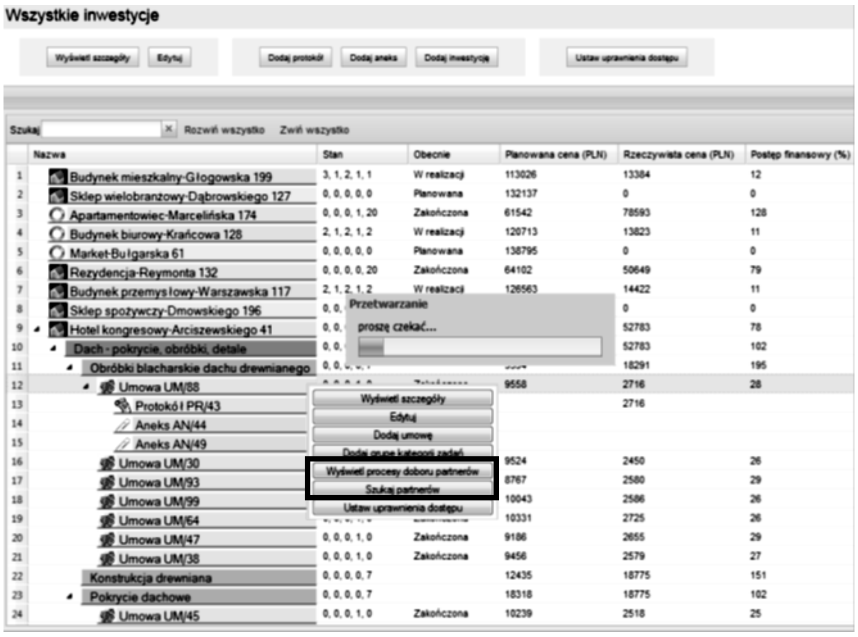
Fig. 6.14. Viewing levels of satisfaction of investment types for the investment

ErGo MatchMaker

Each recommended investment type is an activity pattern. If an activity pattern is on the abstract or the prototype level, the service-oriented summary, the service network schema and the mapping functions of an activity pattern are further used by the *ErGo MatchMaker* application as a set of requirements for the selection of subcontractors. *ErGo MatchMaker* module is an implementation of the RMV method algorithm for instantiation of activity patterns. *ErGo MatchMaker* is closely integrated with *ErGo Organizations* and *ErGo Service* applications (selection and validation of organizations and services against classes of service entities), *ErGo ServiceNet* module (validation of service requirements), *ErGo KPI* (definition of KPIs), *ErGo Investment Types* (access to information concerning recommended investment types) and *ErGo Investments* (launching the method and presentation of the results).

In *ErGo MatchMaker* subcontractors are selected in a collaborative way. Users select organizations that can perform required activities, and then aggregates the chosen organizations. Both single organizations and groups of organizations are discussed with other collaborators. Internal discussions concern potential subcontractors and the terms of the contracts to be negotiated with selected organizations. Internal discussions are followed by external discussions in which the representatives of selected organizations are involved. The goal of the external discussion is the negotiation of the final terms of the contract. When the external discussion ends with a satisfying compromise, the contract is signed and its representation is added to the *ErGo Investments* application.

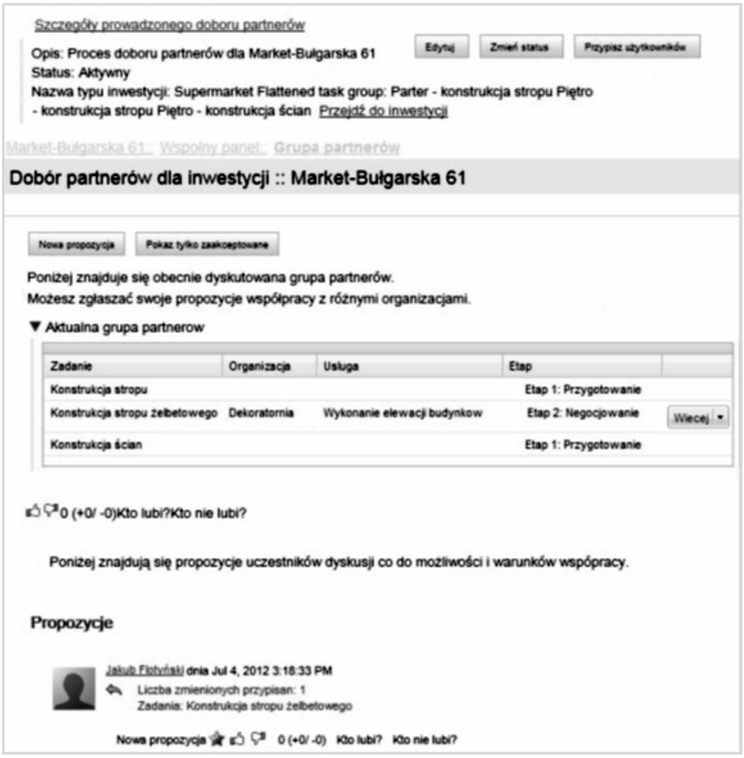
Launching selection of subcontractors in the *ErGo Investment* application is presented in Fig. 6.15 (1).



	Nazwa	Stan	Obecnie	Planowana cena (PLN)	Rzeczywista cena (PLN)	Postęp finansowy (%)
1	Budynek mieszkalny-Głogowska 199	3, 1, 2, 1, 1	W realizacji	113026	13384	12
2	Sklep wielobranżowy-Dąbrowskiego 127	0, 0, 0, 0, 0	Planowana	132137	0	0
3	Apartamentowiec-Marcelińska 174	0, 0, 0, 1, 20	Zakończona	61542	78593	128
4	Budynek biurowy-Krańcowa 128	2, 1, 2, 1, 2	W realizacji	120713	13823	11
5	Market-Bułgarska 61	0, 0, 0, 0, 0	Planowana	138795	0	0
6	Rezydencja-Reymonta 132	0, 0, 0, 0, 20	Zakończona	64102	50649	79
7	Budynek przemysłowy-Warszawska 117	2, 1, 2, 1, 2	W realizacji	126563	14422	11
8	Sklep spożywczy-Dmowskiego 196	0, 0,			0	0
9	Hotel kongresowy-Arciszewskiego 41	0, 0,			52783	78
10	Dach - pokrycie, obróbki, detale	0, 0,			52783	102
11	Obróbki blacharskie dachu drewnianego	0, 0,			18291	195
12	Umowa UM/88			9558	2716	28
13	Protokół PR/43				2716	
14	Aneks AN/44					
15	Aneks AN/49					
16	Umowa UM/30			9524	2450	26
17	Umowa UM/93			8767	2580	29
18	Umowa UM/99			10043	2586	26
19	Umowa UM/64			10331	2725	26
20	Umowa UM/47	0, 0, 0, 1, 0	Zakończona	9186	2655	29
21	Umowa UM/38	0, 0, 0, 1, 0	Zakończona	9456	2579	27
22	Konstrukcja drewniana	0, 0, 0, 0, 7		12435	18775	151
23	Pokrycie dachowe	0, 0, 0, 0, 7		18318	18775	102
24	Umowa UM/45	0, 0, 0, 1, 0	Zakończona	10239	2518	25

Fig. 6.15. Launching selection of subcontractors from *ErGo* Investments

A screen capture of the *ErGo MatchMaker* application is presented in Fig. 6.16. The presented panel permits users to discuss about a group of potential subcontractors. On the top of the panel, the description of the group of activities to be contracted and the associated development process are presented. In the middle of the panel, the table presents the activities to be contracted and the organizations that are proposed as potential subcontractors. In Fig. 6.16, only one organization, i.e., "Dekoratornia", has been proposed as a potential subcontractor for the building of a reinforced concrete slab *("Konstrukcja stropu żelbetenowego"* in Fig. 6.16). Negotiations are ongoing with selected organizations. At the bottom of the panel, another proposition submitted by Jakub Flotyński is presented.



Szczegóły prowadzonego doboru partnerów

Opis: Proces doboru partnerów dla Market-Bułgarska 61
Status: Aktywny
Nazwa typu inwestycji: Supermarket Flattened task group: Parter - konstrukcja stropu Piętro - konstrukcja stropu Piętro - konstrukcja ścian Przejdź do inwestycji

Edytuj
Zmień status
Przypisz użytkowników
Market-Bułgarska 61:: Wspolny panel:: Grupa partnerów

Dobór partnerów dla inwestycji :: Market-Bułgarska 61

Nowa propozycja
Pokaz tylko zaakceptowane

Poniżej znajduje się obecnie dyskutowana grupa partnerów.
Możesz zgłaszać swoje propozycje współpracy z różnymi organizacjami.

▼ Aktualna grupa partnerow

Zadanie	Organizacja	Usluga	Etap	
Konstrukcja stropu			Etap 1: Przygotowanie	
Konstrukcja stropu żelbetowego	Dekoratornia	Wykonanie elewacji budynkow	Etap 2: Negocjowanie	Wiecej ▾
Konstrukcja ścian			Etap 1: Przygotowanie	

0 (+0/ -0)Kto lubi?Kto nie lubi?

Poniżej znajdują się propozycje uczestników dyskusji co do możliwości i warunków współpracy.

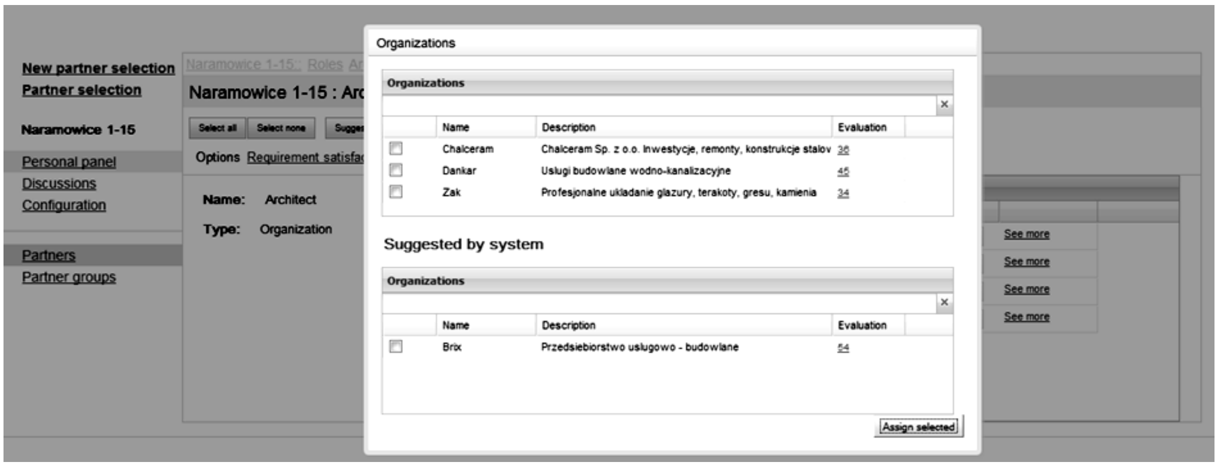
Propozycje

Jakub Flotyński dnia Jul 4, 2012 3:18:33 PM
Liczba zmienionych przypisan: 1
Zadania: Konstrukcja stropu żelbetowego

Nowa propozycja 0 (+0/ -0) Kto lubi? Kto nie lubi?

Fig. 6.16. *ErGo MatchMaker*

In the *ErGo MatchMaker* application, the subcontractor selection process is based on the set of requirements defined in the investment type. As an example, an investment type for the construction of residential buildings defines requirements concerning an architect, electricians, and their relations. These requirements are captured in the service network schema. The *ErGo MatchMaker* uses these requirements together with information coming from the *ErGo Organizations*, *ErGo Services* and *ErGo ServiceNet* modules. As a result, it evaluates suitability of service entities to classes of service entities. The set of organizations best fitting classes of service entities together with evaluation of their conformance is presented in Fig. 6.17. The *ErGo* system also evaluates the full set of organizations in terms of conformance to service requirements (Fig. 6.18).



Name: Architect
Type: Organization

Organizations

Organizations

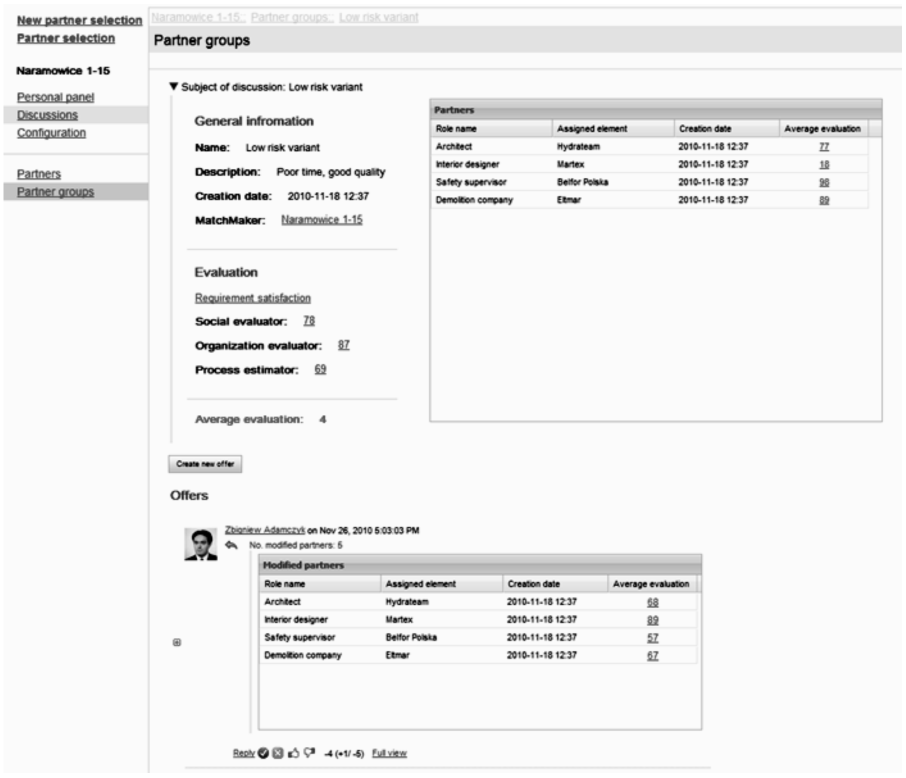
	Name	Description	Evaluation
☐	Chalceram	Chalceram Sp. z o.o. Inwestycje, remonty, konstrukcje stalov	36
☐	Dankar	Uslugi budowlane wodno-kanalizacyjne	45
☐	Zak	Profesjonalne ukladanie glazury, terakoty, gresu, kamienia	34

Suggested by system

Organizations

	Name	Description	Evaluation
☐	Brix	Przedsiebiorstwo uslugowo - budowlane	54

Fig. 6.17. Supported evaluation of assignment of a service entity to a class of service entity



Partner groups

▼ Subject of discussion: Low risk variant

General infromation

Name: Low risk variant

Description: Poor time, good quality

Creation date: 2010-11-18 12:37

MatchMaker: Naramowice 1-15

Evaluation

Requirement satisfaction

Social evaluator: 78

Organization evaluator: 87

Process estimator: 69

Average evaluation: 4

Partners

Role name	Assigned element	Creation date	Average evaluation
Architect	Hydrateam	2010-11-18 12:37	77
Interior designer	Martex	2010-11-18 12:37	18
Safety supervisor	Belfor Polska	2010-11-18 12:37	98
Demolition company	Eltmar	2010-11-18 12:37	89

Offers

Zbigniew Adamczyk on Nov 26, 2010 5:03:03 PM

No. modified partners: 5

Modified partners

Role name	Assigned element	Creation date	Average evaluation
Architect	Hydrateam	2010-11-18 12:37	68
Interior designer	Martex	2010-11-18 12:37	89
Safety supervisor	Belfor Polska	2010-11-18 12:37	57
Demolition company	Eltmar	2010-11-18 12:37	67

-4 (+1/ -5)

Fig. 6.18. Supported evaluation of a VO variant

6.5. RMV Method Real Case Evaluation

The real case evaluation presented in this section encompasses activity sequence pattern discovery, activity pattern identification and recommendation formulation.

In this section, first the analyzed company and its delivery process are shortly described. The name of the company has been anonymized, so it is named Epsilon. Second, the set of events recorded during delivery process instance executions is described. Events are recorded by the warehouse management system. Initial exploration of events is used to provide more detailed characteristics of the delivery process. It is shown that the delivery process shares characteristics of the VO collaborative process. Then, the RMV method is applied to the analysis of the event log. Activity pattern discovery and recommendation formulation is presented for various parameter sets. The section is concluded with presentation of recommendations that were given to Epsilon.

Epsilon company

Epsilon is a production company specializing in production of some physical items exported to Western Europe. Its production volume is over 2 million per year. Epsilon employs more than five hundred people.

The process mining project was conducted jointly with the Epsilon team that was led by the Distribution and Warehouse Manager. The aim of the project was to provide insight into warehouse processes performed in the company. The operation of the company's warehouses is supported by the Warehouse Management System (WMS). The WMS is used by both storekeepers and management:

- The set of storekeepers' activities include: delivery taking, organizing shipment, transporting materials to production, and receiving final products from inventory; the WMS records data associated with all the activities performed by storekeepers;
- Management staff uses WMS to monitor stock levels and supervise storekeepers work.

The WMS supports two main processes performed in the warehouse:

1. *Delivery process* – encompasses activities required to unpack trucks coming from material suppliers, to group materials into pallets, to transport pallets from a delivery line to the warehouse and to the production lines;
2. *Shipment process* – encompasses activities required to take the final product from the production lines to the warehouse and send them to the client or to an external warehouse.

The prototype implementation of the RMV method was validated based on data collected in the WMS during the execution delivery process instances.

In the warehouse, all the materials are organized in pallets. A pallet is the smallest storage and transport unit. Pallets are stored in the warehouse before being forwarded to the production lines. One delivery process instance refers to a number of created and transported pallets and a number of various materials. Materials are categorized into families, which are material types. There are 80 various material families.

Pallets are transported among delivery lines, the warehouse and production lines by 60 storekeepers and quality department employees. The work is conducted on three shifts, 24 hours per day except weekends. Warehouse employees are divided into three not disjunctive groups:

1. *Unpacking group* – responsible for unpacking incoming trucks and moving pallets to the warehouse;
2. *Production group* – responsible for delivery of pallets from the warehouse to the production lines;
3. *Quality group* – responsible for validation of materials quality.

Assignment of employees to groups changes from a shift to another. A person that is not assigned to a particular group should not performed activities assigned to that group, e.g., storekeepers assigned only to the production group should not unpack incoming trucks.

The use of the WMS in warehouse management requires an employee to use mobile scanner to scan the bar code available on the pallet before performing any activity referring to the pallet. The WMS keeps track of pallet life cycle and associates each scanned bar code with an appropriate activity. For instance, once the pallet is scanned in delivery (*Pallet in delivery* activity) the next recorded activity instance must be *Pallet on fork* and finally *Pallet resting*. This sequence refers to moving a pallet from a delivery line to the warehouse. Activities are performed also using the WMS web interface, e.g., *Delivery volume change* activity can be performed only using web interface.

Various activity instances require storekeepers to use various services provided by the WMS. Services are provided by various WMS modules. Some services available in the WMS are provided by more than one WMS module, e.g., delivery details are retrieved from both *Delivery* module and *Purchase* module. Redundant functionality is a consequence of ongoing integration of various systems used in the company.

The process involves 17 activities:

1. *Pallet on fork* – transporting a pallet among the delivery lines, the warehouse and the production lines;
2. *Pallet resting* – putting a pallet in the storage area in the warehouse;
3. *Pallet created* – registering a new pallet in the WMS;
4. *Quality check* – verification of materials quality; typically each delivery includes many material families; for each material family, one pallet is randomly selected to be inspected by the quality group;
5. *Pallet on production* – putting a pallet on a production line;
6. *Delivery confirmation document* – printing confirmation of delivery and giving it to a supplier;
7. *Label printing* – printing a set of labels with bar codes for marking the pallets created during the delivery;
8. *Pallet update* – updating information about a pallet in the WMS;
9. *Delivery close* – truck unpacking is completed; the delivery status in the WMS is updated to *Closed*;
10. *Delivery start* – truck unpacking is started; the delivery status in the WMS is updated to *Started*; all the created pallets are automatically assigned to the started delivery process instance;
11. *Pallet state: repack* – repacking materials on the pallets; this activity is performed if a pallet is destroyed or must be divided into smaller pallets;
12. *Pallet state: banned* – quality group forbids passing a pallet to the production lines;
13. *Pallet state: ok* – pallet quality is set to *Satisfactory* by the quality group; the activity is performed once for each material family; note that one pallet per material family is inspected, but information about the positive result of quality evaluation is stored separately for each pallet from a delivery as a value of *QualityControlResults* pallet attribute (*cf.* Tab. 6.6);
14. *Delivery volume change* – changing the volume of the delivery; the activity is executed if the volume of the delivery is different from the planned volume;
15. *Pallet state: open* – pallet is marked as partially opened; the activity is performed if the pallet has not been fully used on a production line and it is returned to the warehouse as being partially used;
16. *Pallet in delivery* – transporting the pallet to a delivery section in the warehouse;
17. *Pallet deleted* – removing the pallet from the WMS.

Note that while activities *Pallet on fork, Pallet resting, Pallet created, Pallet on production, Pallet update, Pallet state*, Pallet in delivery,* and *Pallet deleted* refer to one particular pallet, other activities are associated with a delivery process instance. Each activity instance is associated with one event in the WMS event log having the same name.

The *de jure* model describing expected execution of each delivery process instance is presented in Fig. 6.19. The process is divided into five phases. In phase 1, the delivery process instance starts with *Delivery start* followed by *Label printing*. In phase 2, materials are divided into pallets. The *Pallet created* activity instance is executed for each pallet. Then each pallet is transported (*Pallet on fork* activity instance) and placed in the warehouse (*Pallet resting* activity instance). After all the pallets are moved to the warehouse, the delivery is closed in phase 3. Two activity instances are executed in this phase: *Delivery close* and *Delivery confirmation document*. In phase 4, one pallet per material family is inspected in term of quality (*Quality check* activity instance). Activity instance *Pallet state: ok* is executed to positively acknowledge quality of pallets with the inspected material family. In phase 5, pallets are gradually transferred to the production lines by performing *Pallet on fork* and *Pallet on production* activity instances. If materials from the pallet are not used at once for production, the state of the pallet is set to open (*Pallet state: open* activity instance), and the pallet is taken to the warehouse (*Pallet on fork* and *Pallet resting* activity instances) to be reused in future. There are two conditions of delivery process instance completion: (1) *Delivery confirmation document* is printed, and (2) all the pallets created in phase 2 are fully utilized in production.

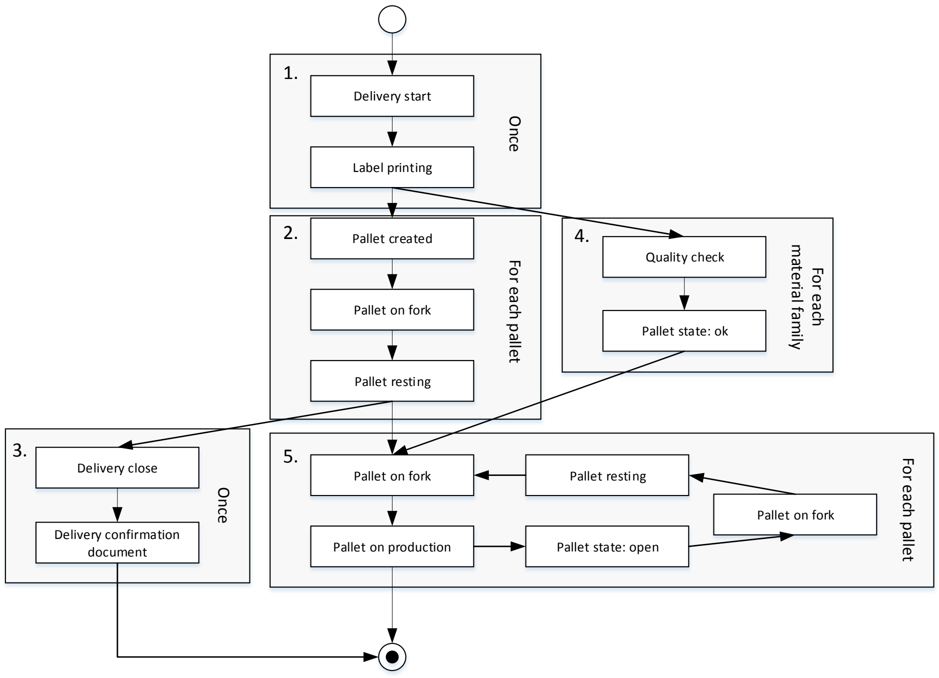


Fig. 6.19. Expected execution of the delivery process instance

Note that six activities are not presented in Fig. 6.19: *Pallet update*, *Delivery volume change*, *Pallet in delivery*, *Pallet state: repack*, *Pallet state: banned* and *Pallet deleted*. These activities do not appear in the proper execution of a delivery process instance. Nevertheless, they can appear at any stage of an improper or exceptional delivery process instance execution.

Analysis scope

Epsilon company deals with the problem of low efficiency of a warehouse operation, where the efficiency is measured by:

1. Percentage of pallets of materials inspected by the *quality* group – it is unacceptable that some materials are forwarded to production either without quality check or with multiple unnecessary quality checks; the cases of low quality of materials must be reported to suppliers as complaints;
2. Number of pallets damaged during truck unpacking or transport of pallets to production lines – this number should be as low as possible; in case of pallet damage, the damaged pallet should be handled according to a special procedure;
3. The amount of time needed to unpack a truck and the average time that pallets with materials are stored in the warehouse – the expected times are provided in the warehouse procedure;

4. Conformance with good practices formulated by the management staff based on long time experience and observation of the warehouse operation – each nonconformance is perceived as an error in storekeeper work.

To increase efficiency of warehouse operation, Epsilon aims at identification of storekeepers, storekeeper groups, shifts and their practices that influence efficiency of warehouse operation in both positive and negative ways. Such identification would be used to eliminate inefficient teams and behaviors and to promote good ones.

To limit the scope of the analysis, a set of questions concerning the delivery process was asked by the Distribution and Warehouse Manager. Questions concerning five key aspects of the delivery process are presented in Tab. 6.5.

Tab. 6.5. Five key aspects of the delivery process

l.p.	Aspect	Questions
1.	Quality assurance	• How to minimize the time and number of activities required to handle complaints?
2.	Pallet damage	• How to minimize the number of damaged pallets? • Which employees are involved in a higher number of damaged pallets? • How to reduce the number of printed and unused labels?
3.	Process performance	• How to lower the time required to unpacking a truck? • How to reduce the overall delivery process instance execution time?
4.	Work distribution	• Which shifts should be involved in execution of particular activities? • Should work be distributed equally among shifts or should shifts be specialized in some sort of activities?
5.	Conformance to the *de jure* model	• What are frequent practices that do not follow the *de jure* model?

The analysis was performed by applying the RMV method to the WMS event log. As mentioned in Section 5.6, a right set of parameters for both activity pattern discovery and recommendation formulation must be selected basing on knowledge concerning: (1) business domain, and (2) basic characteristics of the data set identified at the earlier phases of the method. Thus, the analysis was performed in close collaboration with the Distribution and Warehouse Manager.

The search for relevant activity patterns was done using the *if-cause* approach, i.e., assumptions concerning the RMV method parameters were provided by the Distribution and Warehouse Manager. As a result, the RMV method was applied to the event log multiple times using various parameter sets. The application of the RMV method to the event log with one particular set of parameters is further referred to as an *experiment*. Each experiment led to identification of different activity patterns and formulation of recommendations. The most relevant modifications in parameter sets included various definitions of: context class function, context distance function, functions mapping *recom_index* and *nonrecom_index* values and required activity patterns lengths.

WMS event log

The set of attributes associated with events in the WMS event log is presented in Tab. 6.6. The *AttributeName* column presents event attribute names. Description of each attribute is given in the *Description* column. The *Attr.Type* column informs if an attribute is associated either with a service provider, or a service consumer, or the service interface (*cf.* Section 5.4.3). For clarity, example values of all the attributes are presented in the *Domain example* column. Not all the events have the same set of attributes. The set of events sharing a particular attribute is presented in the *Events* column, e.g., *PalletNo* attribute is stored only for events referring to a particular pallet being created, updated, moved, etc.

There are 18 attributes used in various phased of the RMV method. Note that not all attributes are used in the same manner:

- Attributes *DeliveryID* and *ActivityInstanceID* are used for grouping events into delivery process instances; attributes are not associated with any service entity;
- *Timestamp* and *Trans* attributes are used only to determine the temporal relations among activities represented by events; attributes are not associated with any service entity;
- Remaining fourteen attributes are used as attributes of activity instance descriptions during activity pattern discovery.

Attributes accounting for event context are presented in Tab. 6.7. Note that *Shift* and *Warehouse* event attributes are used both as context attributes and regular event attributes.

The set of attributes stored in the WMS event log, which describe each delivery process instance, is presented in Tab. 6.8. Values of *Resource*, *AdminName*, *SupplierName*, *PalletArticles* are anonymized. Also final product names are anonymized.

Tab. 6.6. Event attributes

l.p.	Attribute name	Description	Events	Attr. type	Domain example
1.	*DeliveryID*	Delivery process instance identifier	All	-	*DS0000007337*
2.	*ActivityInstanceID*	Activity instance identifier	All	SI	*2435*
3.	*Activity*	Activity name	All	SI	*Pallet on fork, Delivery close*
4.	*Resource*	Storekeeper name	All	SC	*mman, mrob*
5.	*ModuleName*	WMS module name	All	SP, social-SI	*Obsolete, Purchase, Control, Management, Delivery, Inventory, Complaint*
6.	*Trans*	Event type (*cf.* Section 5.4.3)	All	-	*complete*
7.	*Group*	Stakeholder group	All	social-SC	*Unpacking, Production, Both (Unpacking and Production), Quality*
8.	*Shift*	Stakeholder shift	All	social-SC	*Morning, Afternoon, Night*
9.	*Timestamp*	Time of the activity instance execution	All	SI	*2013/01/02 08:55:27.247*
10.	*PalletNo*	Pallet identifier	*Pallet in delivery, Pallet on fork, Pallet resting, Pallet created, Pallet on production, Pallet update, Pallet state: *, Pallet deleted*	service Entity-SI	*799000000001559562*
11.	*PalletArticles*	Material families on pallet	*Pallet in delivery, Pallet on fork, Pallet resting, Pallet created, Pallet on production, Pallet update, Pallet state: *, Pallet deleted*	service Entity-SI	*M1, M2*
12.	*Warehouse*	Warehouse name	All	service Entity-SI	*Base warehouse, Warehouse in ddp Ociąż., Warehouse sewing*
13.	*Recommendation Done*	Information whether the recommendation generate by the WMS concerning storage place for the pallet was followed	*Pallet resting, Pallet on production*	service Entity-SI	*True, False*
14.	*QualityControl Results*	Approved in terms of quality	*Quality check*	service Entity-SI	*1; 0*
15.	*Damaged*	Pallet put in the storage area reserved for the damaged pallets	*Pallet resting*	service Entity-SP	*1; 0*
16.	*System*	Name of the system	All	service Entity-SP	*WMS*
17.	*AdminName*	Name of system admin on the time	All	service	*krys, phan*

		of activity instance execution		Entity-SP	
18.	*Privilages*	Security rights assigned to user	All	social-SP	manager

Tab. 6.7. Context attributes of activity instances

l.p.	Attribute name	Description	Activities	Domain example
1.	*Month*	Name of the month	All	*March*
2.	*StockLevel*	Warehouse stock level in percent	Pallet created, Pallet deleted, Pallet on production	*71.87%*
3.	*Shift*	Stakeholder shift	All	*Morning, Afternoon, Night*
4.	*Warehouse*	Warehouse name	All	*Base warehouse, Warehouse in DDP, Warehouse sewing, Production-matrasses, Production-sewing, Quarantine*

Tab. 6.8. Delivery process instance attributes

l.p.	Attribute name	Description	Domain example
1.	*ProcessResult*	Delivery process instance output; the delivery process instance following *de jure* process from Fig. 6.19 has output *OK*; other possible outputs are: *Complaint, Deleted, Disposal*	*OK, Deleted, Disposal, Complaint*
2.	*PurchaseValueDPS*	The monetary value of the delivery in PLN	*500 000, 320 000*
3.	*ArticlesOrdered*	Ordered materials with quantity in format: material family/quantity	*A1/80, A2 16002000/112*
4.	*ArticleDelivered*	Delivered materials	*M1, M2*
5.	*ArticleVolume*	Quantity of delivered materials	*80; 112*
6.	*SupplierName*	Supplier name	*A; B*
8.	*OrderMonth*	Delivery month	*March*
10.	*NoDamagedPallets*	Number of pallets damaged during the delivery process instance execution	*0; 11.5*
11.	*PalletVsLabelQty*	Depending on percentage of printed labels that were used with pallets, use of labels is categorized into five categories	*Full, Fine, Medium, Small, Unacceptable*
12.	*LablesToErase*	Number of pallet labels to utilize	*8*

Delivery process analysis was performed on the basis of 152 523 events associated with 1448 delivery process instances performed during 6 months of the warehouse operation. During this time, 26 641 pallets were delivered to the warehouse and forwarded to the production lines. The average duration of a delivery process instance was 4 days 17 hours 45 minutes.

The delivery process has spaghetti-like structure (*cf.* Section 2.1). 1648 delivery process instances were executed according to 1590 variants. This means that execution of a process instance is not fully determined and almost each process instance is executed in a unique way. Distribution of numbers of events among variants is presented in Fig. 6.20[13]. Values on the horizontal axis represent the number of events. Values on the vertical axis represent number of delivery process instances. Average number of events per delivery process instance is 105, but 46% of delivery process instances have less than 50 events.

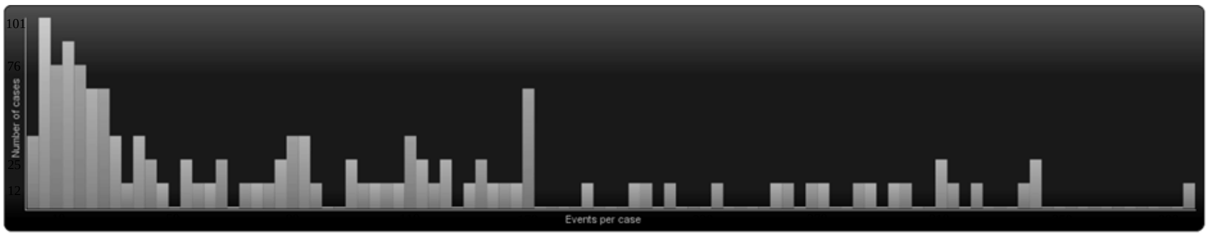


Fig. 6.20. Distribution of number of events among process instances

Complexity of the delivery process is confirmed by the process presented in Fig. 6.21[14] – rectangles represent activities, arrows represent temporal relations among activities. The complicated structure of the process map proves its spaghetti-like character. The process map presented in Fig. 6.22 is yet more complex. Here each rectangle corresponds to an activity performed by a particular storekeeper using a particular WMS service – the number of rectangles is much bigger, i.e., there is a large diversity in assignment of actors and WMS system modules to each activity instance.

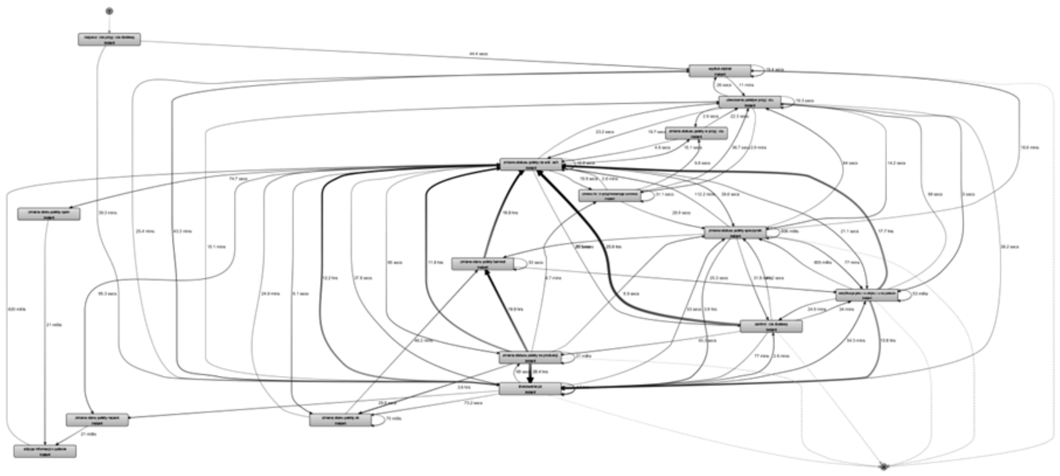


Fig. 6.21. Process map: rectangle corresponds to activity

[13] The figure is generated using Fluxicon Disco software, http://www.fluxicon.com/disco/
[14] The figure is generated using Fluxicon Disco software, http://www.fluxicon.com/disco/

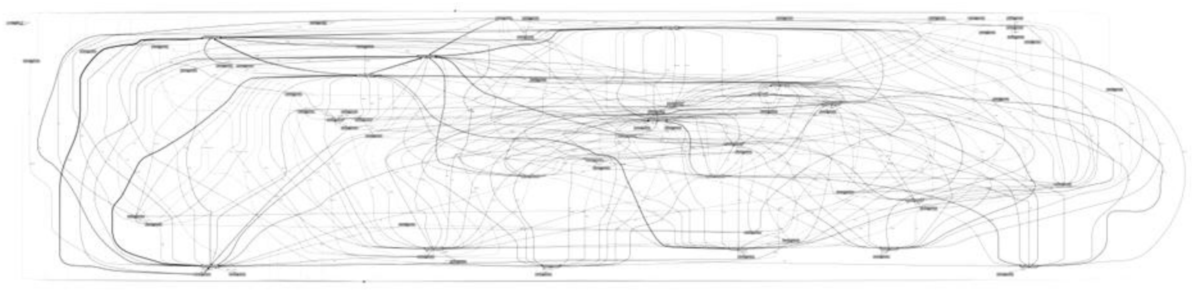


Fig. 6.22. Process map: rectangle corresponds to a triple: activity name, storekeeper, WMS module

Execution of all the recorded delivery process instances involved 55 persons and 8 WMS modules. Frequency of activities recorded in the WMS event log is presented in Tab. 6.9.

Tab. 6.9. Frequency of activities in the event log

l.p.	Value	Frequency	Relative frequency
1.	Pallet on fork	48077	31.52%
2.	Pallet resting	30816	20.20%
3.	Pallet created	26806	17.58%
4.	Quality check	17428	11.43%
5.	Pallet on production	15614	10.24%
6.	Delivery confirmation document	3934	2.58%
7.	Label printing	3187	2.09%
8.	Pallet update	1540	1.01%
9.	Delivery close	1448	0.95%
10.	Delivery start	1448	0.95%
11.	Pallet state: repack	1433	0.94%
12.	Pallet state: banned	274	0.18%
13.	Pallet state: ok	152	0.10%
14.	Delivery volume change	107	0.07%
15.	Pallet state: open	107	0.07%
16.	Pallet in delivery	76	0.05%
17.	Pallet deleted	76	0.05%

Despite the fact that the delivery process is performed within one organization, the organization of the warehouse operation follows the characteristics of VO (*cf.* Sections 2.1, 2.2, and 5.4.1):

- The set of VO members includes: storekeepers, quality department employees, groups and shifts of storekeepers and quality department employees;
- The set of service interface descriptions used by the VO members is the set of services provided by the WMS system modules that are interchangeably used during execution of activities;
- The set of relations among the VO members encompasses: (1) relations among people, i.e., relations of belonging to a particular *day*, *afternoon* or *night* shift, and relations of belonging to the *unpacking*, *production* or *quality* group; (2) relations among services, i.e., relations of being provided by the same WMS module;

- Collaboration is guided by the delivery process having all the characteristics of a VO collaborative process:
    - Large diversity in assignments of storekeepers and WMS module services to activity instances;
    - Unstructured character of the delivery process;
    - The set of storekeepers and services as well as their roles change during delivery process instance executions;
    - Similar instances of a delivery process are interrelated;
    - The delivery process has unpredictable and emergent character.
- Virtual organization members are largely autonomous, separately managed, differently situated in the Epsilon organizational structure, have heterogeneous culture, social capital and goals;
- Collaboration within VO is performed on demand of the Production Department that is a VO client.

Thus, the problem of identification of storekeepers, storekeeper groups and shifts, investigated by Epsilon that would increase efficiency of delivery process is indeed the problem of partner and service selection for VO. Consequently Epsilon company, like SOVOBE, provides technical and organizational infrastructure supporting collaboration of such VO.

Experiments

Further in this section, four experiments are presented. *Experiments A* and *B* refer to discovery of activity patterns. *Experiments C* and *D* refer to recommendation formulation. This representative set of experiments permits to analyze the impact of parameter values on the quantity of results generated by the RMV method. As an example, one of discovered activity patterns is described in detail. The section is concluded with the list of business-level recommendations formulated by interpretation of identified activity patterns. These recommendations address questions from Tab. 6.5. They were successfully applied by the production company Epsilon to improve the delivery process.

Experiments A and B

Parameter values used in Experiment *A* and Experiment *B* are presented in Tab. 6.10. The parameter sets are subsets of parameters presented in Section 5.6. The set of parameters used in Experiment *A* supports discovery of the activity patterns having the largest possible number of activity descriptions. The parameters set *in* Experiment *B* aims at discovery of a set of activity patterns having high diversity of contexts. Discovered activity patterns are then shorter, but they support more parts of delivery process execution.

Values of *Duration constraint*, *Gap constraint* and *Maximum timestamp constraint* parameters are the same in both Experiments *A* and *B*.

Values of other parameters in Experiment *A* state that:

- The analyzed set of activity instance description attributes encompasses all the 14 relevant attributes from Tab. 6.6;
- The minimum number of attributes that must describe an activity in an activity pattern is set to 4; this constraint makes the discovered activity patterns more information rich and meaningful in comparison with activity patterns with a smaller number of

attributes in activity descriptions; this constraint reduces the number of discovered activity patterns;

- Only activity patterns having no less than 15 activity descriptions are considered;
- Presented results are discovered for required support level equal to 0.2;
- No constraint is set on maximum sequence length;
- No constraint is imposed on attribute names that must be present in each activity description from an activity pattern.

In Experiment *B*:

- Maximum sequence length is set to 25; this constraint imposed on maximum sequence length boosts the number of discovered activity patterns;
- Two activity instance attributes are removed from the set of analyzed attributes: *RecommendationDone* and *AdminName* attributes – the set of attributes is limited to 12;
- The set of obligatory attributes for each activity pattern is set to identifiers of service consumer (SC), service provider (SP) and service interface (SI); only activity patterns with defined full set of actors should be discovered;
- The minimum number of activity instance description attributes is increased to 6;
- The required support is increased to 0.25.

Tab. 6.10. User-defined parameters for Experiments A and B

l.p.	Parameter name	Experiment A	Experiment B
	Event attributes		
1.	Scope and types of activity instance description attributes	14	12
2.	Minimum activity instance description length	4	6
3.	Obligatory attributes	-	SI, SP, SC
Constants			
1.	Duration constraint	1 209 600 000 ms (336 hours)	1 209 600 000 ms (336 hours)
2.	Gap constraint	172 800 000 ms (48 hours)	172 800 000 ms (48 hours)
3.	Maximum timestamp constraint	2013/05/26 23:59:00.000	2013/05/26 23:59:00.000
4.	Minimum sequence length constraint	15	15
5.	Maximum sequence length constraint	-	25
6.	Required activity pattern support level	0.2 (290 process instances)	0.25 (262 process instances)

In Experiment *A*, the total number of discovered closed activity patterns is 1079. The maximum support for an activity pattern is 0.33. There are 53 different activity patterns having the maximum support. There are 12 maximum activity patterns with maximum length of 47 activity descriptions. The highest support of maximum activity pattern is 0.2. Average pattern length is 24.8.

In Experiment B, the total number of discovered closed activity patterns is 1360. This number is larger than the number of activity patterns discovered in Experiment *A*, but activity patterns are shorter on average. Average pattern length is now 16.8. The maximum support for activity pattern is 0.3. There are 71 different activity patterns having such support. There are 21 activity patterns having maximum allowed length. The highest support of the maximum activity pattern is 0.24.

Experiments C and D

Values of parameters for Experiments *C* and *D* are presented in Tab. 6.11. In particular, the parameterization of the RMV method in Experiments *C* and *D* included definition of the following functions: context distance functions, function mapping context to context class, function mapping *recom_index* and *nonrecom_index* values. The functions used in the Experiments *C* and *D* are simple but powerful enough to formulate relevant recommendations. In other experiments that are not described in this section due to limited space, many more advanced functions were used to answer Distribution and Warehouse Manager questions.

Tab. 6.11. User-defined parameters for experiment C and D

l.p.	Parameter name	Experiment C	Experiment D
	Event attributes		
1.	Activity instance context scope	All	All
2.	Process instance attributes scope	All, *ProcessResult*	All, *ProcessResult*
	Functions		
1.	Context distance function	Function name: *eval*	Function name: *eval*
2.	Function mapping context to context class	Function name: *context_C*	Function name: *context_D*
3.	*recom_index* and *nonrecom_index* calculation	Function name: *max_dur* Cost attribute: *Timestamp*	Function name: *max_dur* Cost attribute: *Timestamp*
	Constants		
1.	Classification group size	300	100
2.	Delta	300 000 (5 min)	300 000 (5 min)
3.	Minimum accepted sensitivity	30%	30%

In Experiments *C* and *D*, the set of activity instance context attributes includes all the context attributes from Tab. 6.7. The set of attributes describing process instances includes attributes from Tab. 6.8. The name of the outcome attribute is *ProcessResult*.

In Experiments *C* and *D*, the context distance function *eval* is the same. The function compares elements of context class and context (*cf.* Section 5.4.2) using a subfunction for each element. The set of subfunctions for each context element (*PID*, *EC*, *H*, *SE*, *S* – *cf.* Section 5.4.2) is as follows:

- *PID* – if all the context element constraints from a context class referring to attributes of the process instances are satisfied, the subfunction returns 0, otherwise 1;
- *EC* – if at least half of the event context element constraints from a context class referring to event attributes are satisfied, the subfunction returns 0, otherwise 1;
- *H* – if all the events indicated in a context class already appeared in the partial trace of the process instance, the subfunction returns 0, otherwise 1;
- *SE* – if all the constraints concerning the frequency of appearance of actors are satisfied, the subfunction returns 0, otherwise 1;
- *S* – this context element is ignored in the analysis, the subfunction always returns 0.

Values returned by the subfunctions are then summarized. The value of the *eval* function returned for a given context class and the activity pattern context is between 0 and 4, where the smaller the value the more similar context and context class are.

In Experiments *C* and *D*, values of *recom_index* and *nonrecom_index* are calculated based on the *max_dur* function. Function *max_dur* takes a delivery process instance as an argument and returns the time that was required to execute the delivery process instance. Calculation is based on *Timestamp* attributes of each event. Having a set of activity pattern contexts enabled for the context class, the value of *recom_index* is calculated as a sum of *max_dur* function results calculated for all the delivery process instances supporting the activity pattern in the set of enabled activity pattern contexts. Similarly, the value of *nonrecom_index* is calculated as a sum of *max_dur* function results returned for all the delivery process instances that support activity patterns different from the activity pattern in the set of enabled activity pattern contexts.

In Experiments *C* and *D*, the value of *delta* parameter is set to 5 minutes. Minimum accepted sensitivity is set to 70%.

Experiments *C* and *D* are different in terms of functions denoting context class for a given context, and classification group size.

In Experiment *C*, the *context_C* function transforms context of running delivery process instance to a very general context class. In Experiment *C*, the recommendation was requested for the following context *co*:

co ={ *PID* =

$$\left\{\begin{matrix} \langle ProcessResult, Unknown\rangle, \langle PurchaseValueDPS, 150\ 000\rangle,, \\ \langle ArticlesOrdered, M1/90\rangle, \langle ArticlesDelivered, M1\rangle, \\ \langle ArticleVolume, 90\rangle, \langle SupplierName, XXZ\rangle, \langle OrderDate, 2013/02/01\ 08{:}55{:}27.247\rangle,, \\ \langle OrderMonth, February\rangle, \langle LabelSumQty, 36\rangle, \langle NoDamagedPallets, 0\rangle, \\ \langle PalletsVsLabelQty, Full\rangle, \langle LabelsToErase, 0\rangle, \end{matrix}\right\}$$

EC =

$$\left\{\begin{matrix} \langle Warehouse, Base\ Warehouse\rangle, \langle Shift, Morning\rangle, \langle StockLevel, 55\%\rangle, \\ \langle Month, February\rangle \end{matrix}\right\}$$

H =

$$\left\{\begin{matrix} Delivery\ start, Label\ printing, Qualityceck, Pallet\ created, \\ Pallet\ state{:}\ ok, Pallet\ on\ fork, Delivery\ close, Pallet\ resting, \\ Pallet\ created, Pallet\ on\ fork, Pallet\ resting \end{matrix}\right\}$$

SE =

$$\{\langle mmar, 2\rangle, \langle mtom, 5\rangle, \langle kset, 3\rangle\}$$

S =

$$\{Pallet\ resting, Pallet\ created, Pallet\ on\ fork\} \qquad \}$$

The *context_C* function transforms attributes associated with each context element (PID, EC, H, SE, S) to context constraints and maps the context *co* to the following context class $co\_C^{\propto}$:

$co\_C^{\propto}$ ={ *PID* =

$$\left\{\begin{matrix} \langle ProcessResult, = OK\rangle, \langle OrderMonth, \ni \{January, February, March\}\rangle, \\ \langle ArticlesDelivered, \ni \{M1, M2\}\rangle, \langle SupplierName, \ni \{XXZ, XYZ\}\rangle \end{matrix}\right\}$$

EC =

$$\left\{\begin{matrix} \langle Warehouse, = Base\ Warehouse\rangle, \langle Shift, \ni \{Morning, Afternoon, Night\}\rangle, \\ \langle Month, February\rangle, \langle NoDamagedPallets, < 10\rangle \end{matrix}\right\}$$

H = ∅

SE =

$$\{\langle mmar, > 0\rangle, \langle mtom, > 0\rangle, \langle kset, > 0\rangle\}$$

S = ∅ }

Note that during mapping, some attributes of the context elements are ignored and all the context element values are replaced with predicates.

Finally, in Experiment *C*, the classification group size is set to 50, i.e., only activity instances appearing in one of 50 contexts that are the most similar to the context class according to the value of the *eval* function are considered for recommendation.

In Experiment *D*, the function mapping context to context class and classification group size are changed. Function *context_D* maps context *co* to the following context class $co\_D^{\propto}$:

$co\_D^{\propto}$ ={ *PID* =

$$\begin{Bmatrix} \langle ProcessResult, = OK\rangle, \langle OrderMonth, \ni \{January, February\}\rangle, \\ \langle ArticlesDelivered, \ni \{M1\}\rangle, \langle SupplierName, \ni \{XXZ, XYZ\}\rangle, \\ \langle NoDamagedPallets, < 10\rangle \end{Bmatrix}$$

EC =

$$\begin{Bmatrix} \langle Warehouse, = \{Base\ Warehouse\}\rangle, \langle Shift, = \{Morning\}\rangle, \langle StockLevel, > 55\%\rangle, \\ \langle Month, February\rangle \end{Bmatrix}$$

H =

$$\begin{Bmatrix} Delivery\ start, Label\ printing, Qualityceck, Pallet\ created, \\ Pallet\ state{:}\ ok, Pallet\ on\ fork, Delivery\ close, Pallet\ resting, \\ Pallet\ created, Pallet\ on\ fork, Pallet\ resting \end{Bmatrix}$$

SE =

$$\{\langle mmar, > 0\rangle, \langle mtom, > 0\rangle, \langle kset, > 0\rangle\}$$

S = ∅ }

Note that the context class $co\_D^{\propto}$ more precisely defines constraints on activity pattern contexts than the context class $co\_C^{\propto}$ from *Experiment C*. The classification group size is limited to 100. Only activity instances appearing in one of 100 contexts that are the most similar to the context class are considered for recommendation.

To determine the set of enabled activity pattern contexts in both Experiments *C* and *D*, 3022 different activity pattern contexts were compared with the context classes. This means that on average, each activity pattern appears in 6 contexts.

In Experiment *C*, on the basis of values returned by the *context_sim* funtions, 300 activity patterns were classified as enabled. These 300 contexts correspond to 196 activity patterns that were further analyzed for recommendation. The highest number of contexts from the set of enabled contexts that belong to one activity pattern was 25. On the other hand, in the set of enabled contexts, one context was common for 18 enabled activity patterns. From the set of 196 activity patterns, 31 were excluded from being considered for recommendation due to their sensitivity lower than assumed minimum. Maximum recorded values of sensitivity and specificity were 65% and 54% respectively. The minimum recorded weighted context distance was 0.5. The values of *recom_index* and *nonrecom_index* for the activity pattern with the highest sensitivity value were 16 days and 12 days, respectively. This means that it is not worth to follow this activity pattern because *recom_index* is greater than *nonrecom_index*. Thus, the benefit of following this activity pattern is negative. The largest calculated benefit from following an activity pattern, calculated as a difference between *recom_index* and *nonrecom_index*, was: 5 days. The activity pattern with the largest benefit has, however, higher weighted context distance.

In Experiment *D*, the number of activity patterns that appear in the set of 100 enabled contexts is 100. Thus, each enabled activity pattern has a different enabled context, but all the contexts are very similar to each other. This is the result of more strict constraints used in this experiment (*cf.* Tab. 6.11). Minimum recorded weighted context distance is 1,1. In the set of enabled activity patterns, the differences between potential benefits are very small. On the other hand, there is large diversity in sensitivity and specificity values of enabled activity patterns.

Final findings

An example of a discovered activity pattern is presented in Fig. 6.23. The activity pattern was enabled for context class $co\_C^{\propto}$. For the sake of simplicity, relatively small activity pattern is presented in Fig. 6.23, i.e., constraints used during activity pattern discovery concerning minimum sequence length are dropped. The activity pattern encompasses six activity descriptions *I, II, III, IV, V, VI*. Each activity description has one service description assigned. Elements of each service description have classes of service entities assigned coming from the service network schema. Note that not all the elements of service descriptions have classes of service entities assigned. Some classes of service entities define characteristics of service entities, e.g., the class of service interface assigned to the service description of activity description *I* means that a service provider required for execution of this activity must be the WMS system module administrated by *rros*. There is a set of empty classes of service descriptions that do not impose any requirements on service entity attributes but, on the contrary, require some particular service relations, e.g., service consumer from the activity description *I* must be the same unpacking group and day shift as the service consumer from the activity description *II*. Majority of classes of service entities from the service network schema have service entities assigned from the service network. Note that service requirements denoted $SP^{\propto}$ and service relations denoted *SP* indicate *provides*/*is provided by* relation among service interfaces and service providers. Note an interesting case of activity description *III*. Service entities required for execution of activity *III* are unknown. The only known thing about activity *III* is that it follows activity *II*, proceeds activity *IV*, and a service consumer assigned to activity *III* should be in group *Both* as service consumer *kkuj* assigned to activity *V*. The activity pattern has the following credibility characteristics:

- Difference between *recom_index* and *nonrecom_index* values is (*cdif* value): 40 hours (2 days 16 hours);
- Sensitivity: 75%;
- Specificity: 71%;
- Weighted context distance: 0.5;
- Support: 31%;

The activity pattern characteristics confirm that this activity pattern is credible and should be followed.

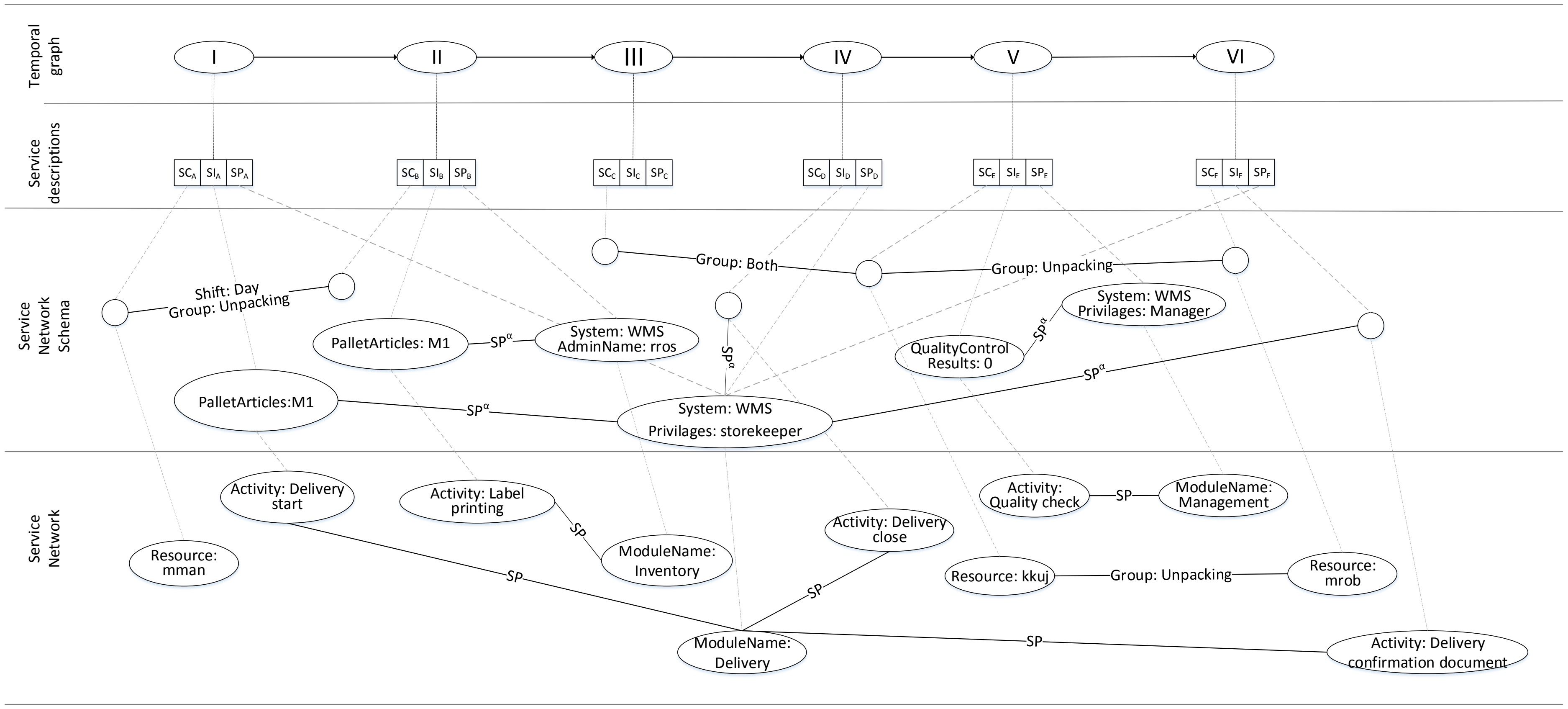


Fig. 6.23. An example of a discovered activity pattern

Identification of the above activity pattern results in the following recommendation:

- If the aim is to limit the time required for the overall delivery process execution and if the delivered materials are *M1*, the process instance execution should be started by *mman* employee using *Delivery start* service provided by the *Inventory* module of the WMS system. The second activity should be executed using *Label printing* service provided by the *Inventory* module. The activity should be followed by some activity performed by the service consumer coming from the same *Both* group as service consumer *kkuj* using *Delivery close* service in execution of activity *V*. For activity *IV*, *Delivery close* service should be provided by the *Delivery* module of the WMS system. Data associated with this activity should be saved in the database using *storekeeper* security role. Activity *IV* should be followed by *Kkuj* service consumer using *Quality check* service provided by the *Management* module to execute this activity. *Kkuj* is in *unpacking* group together with *mrob* service consumer from activity *VI*. Activity *VI* requires the use of *Delivery confirmation document* service provided by the unknown WMS module.

In a similar way other business-level recommendations were formulated and presented to company Epsilon. Note that the above example of recommendation is very detailed. In practice, after the analysis of numerous activity patterns, recommendations are generalized and simplified to be applicable in the warehouse operation, i.e., too detailed recommendations are difficult to comprehend and thus to apply by the Distribution and Warehouse Manager. The subset of generalized recommendations given to company Epsilon is presented in Tab. 6.12.

Tab. 6.12. Recommendations formulated using the RMV method

l.p.	**Aspect**	**Questions**
1.	Quality assurance	• If complaints concern *M2*, *M3*, *M1* materials, it is recommended to involve *msob*, *hwit*, *mmen*, *kgrz* storekeepers to handle these complaints together on a day shift; • If complaints concern *M4*, *M5*, *M6* materials, it is recommended to involve *hwit*, *mmen*, *ptur*, *iszu*, *mbaj*, *mkuj* storekeepers to handle these complaints together on a day shift;
2.	Pallet damage	• It is always recommended not to include *ptur* storekeeper in the production group on the *night* shift; • It is always recommended to limit the number of storekeepers involved in the delivery process instance execution; • If the *Pallet state: banned* activity is executed, it is recommended to execute *Pallet on fork* and *Pallet resting* activities by *mmar* or *mmah* storekeepers, where the pallet is put on storage area reserved for damaged pallets;
3.	Process performance	• If the delivery process instance is executed on a *night* shift, the *Quality check* should be performed after the truck is fully unpacked; moreover, all the pallets should be checked within one shift by people form the *quality* group; • If the delivery process execution is performed on the *night* shift with high stock levels, it is not recommended that activities are executed by storekeepers assigned both to the *unpacking* and *production* groups;

		• It is always recommended to execute phases 3 and 4 of the delivery process in parallel; • It is always recommended to eliminate multiple quality checks performed by the quality group throughout the delivery process instance execution; • It is always recommended to eliminate delivery confirmation document printing at the early stages of delivery process instance execution; • It is always recommended to eliminate multiple executions of the *Delivery close* activity within one process instance; • It is always recommended to perform *Quality check* of all the materials before moving any pallet to a production line;
4.	Work distribution	• If the delivery process instance is executed on an *afternoon* or *night* shift and if the delivery concerns materials used in production of product type *X*, it is recommended not to put *jsaw* and *mkuj* storekeepers on the same shift; • There is an unwanted frequent activity pattern including the following sequence of activities: *Pallet created*, *Pallet on fork*, *Pallet resting*, *Pallet on fork*, *Pallet in delivery*, *Pallet on fork*, *Pallet rest*; this pattern indicates that the pallet is unnecessarily taken from a delivery line to the warehouse and later returned to the delivery line to be finally transported to the warehouse for the second time; the pattern appears independently from the context.
5.	Conformance to *de jure* model	• Unwanted execution of the *Delivery volume change* activity typically takes place if the delivery is associated with *WZX* supplier and the number of *LabelsToErase* is greater than zero; the *Delivery volume change* activity is typically performed after the *Quality Check* activity.

Note that recommendations include aspects concerning service relations that should appear among storekeepers (e.g., assignment to a group or shift), characteristics of actors (e.g., material families transported on pallets), set of activities to be executed, assignment of actors to activities, information concerning process instance characteristics, activities proceeding activity patterns execution and finally potential benefit of following the recommendations. Note also that some discovered activity patterns and formulated recommendations are not limited to a particular context.

Discovered activity patterns and formulated contextual recommendation helped the Epsilon company in identification of storekeepers, storekeeper groups, shifts and their practices that increase or reduce efficiency of the warehouse operation. RMV method results were used in further investigation of reasons explaining the appearance of indicated activity patterns. This led to a number of actions undertaken by the company including: reassignment of responsibilities among warehouse employees, changes is distribution of work among shifts, trainings of storekeepers and quality department employees, and WMS system re-configuration.

7. Conclusions

The *Recommendation Method for Virtual Organizations* RMV presented in this dissertation provides a solution to the problem of computer support for unstructured, emergent and unpredictable VO collaborative processes through recommendations of activity patterns based on contexts.

Four ideas are the basis of the RMV method. First, VO collaborative event logs contain information about interactions among collaborators that appear during executions of various VO collaborative process instances. Second, contexts influence behavior of collaborators. Third, frequently repeatable collaborators' behavioral patterns, called activity patterns, can be discovered through analysis of data stored in the VO collaborative events logs. Forth, discovered activity patterns can be evaluated as good or bad practices and then used in other instances of VO collaborative process instances to improve their efficiency.

The RMV method is composed of four parts: activity sequence pattern discovery method, activity pattern identification method, recommendation formulation method and activity pattern instantiation method. Each of these methods offers a value by itself, but combined together constitute an approach that provides efficient support for execution of VO collaborative process instances and satisfies nine requirements for computer support for VO collaborative processes presented in Section 5.1: (1) guidance for process instance execution, (2) support for conformance analysis, (3) support for adaptation and flexibility, (4) descriptive model, (5) computer supported approach, (6) collaborative wisdom, (7) reusability, (8) social aspect and context, and (9) continuous instantiation.

The RMV method maximizes the scope of guidance for VO collaborative process execution by recommendation of activity patterns suitable for the current context of VO collaborative process (requirement 1). Though the RMV method generates a list of best matching activity patterns, the final selection of an activity pattern to be included in VO collaborative process instance execution is up to a group of selecting collaborators (requirement 5). Activity patterns include not only specification of partially ordered set of activities to be performed as the next ones in the VO collaborative process instance, but also indicates expected relations among service entities, their characteristics and assignment to activities (requirement 8).

A recommendation may be requested at any moment of VO collaborative process instance execution. The recommendation concerns the set of activities suitable for the current context. Once context changes and a new recommendation is requested, the set of presented recommendations is in general different (requirement 8). Such context-awareness supports the adaptation of the VO collaborative processes to changing environment. Due to context awareness and the fact that activity patterns encompass a set of activities to be performed in the nearest future, the RMV method allows collaborators to change planned set of activities if it is required by a context change. Thus, the RMV method offers user guidance at little cost of flexibility of a VO collaborative processes instance execution (requirement 3).

Recommended activity patterns are descriptive, not prescriptive (requirement 4), i.e., activity patterns do not represent assumptions concerning collaborators’ behavior, but capture real and actual repeatable patterns of collaborators’ behavior. Activity patterns discovered in one VO collaborative process instance can be reused in other VO collaborative process instances by other collaborators (requirement 6 and 7). Recommendation mechanism encompasses recommendation monitoring which validates the willingness of collaborators to follow recommendations (requirement 2). Outcomes of recommendations monitoring are used during formulation of the next recommendations. Activity patterns that had been recommended to the group of collaborators, but were not followed by the group, are excluded from the future recommendations. Similarly, if monitoring captures the fact that the group of collaborators consequently follows activity patterns discovered in VO collaborative process instances executed without their participation, such activity patterns are promoted during recommendation formulation (requirement 6).

Finally, the RMV method supports continuous instantiation of VO collaborative processes such that the selection of service entities is performed throughout the VO collaborative process instance execution every time a particular activity patterns is selected to be followed (requirement 9). Selection of collaborators and services is based on criteria relevant to collaborative processes including collaborator and service features and service requirements (requirement 8).

The RMV method goes beyond the existing methods of recommendation based on process mining by providing recommendations for processes that are unstructured, emerging and unpredictable. Moreover, recommendation does not encompass information only about one activity that should be executed as the next one, but it encompasses a set of partially ordered activities enriched with information concerning desired collaborators, their features and character of service relations among them.

The application of the RMV method to the real case data presented in Section 6.5 shows that the RMV method permits formulation of non-trivial, accurate recommendation that are very relevant for a given business context and a particular VO collaborative process. Classes of service entities and service requirements discovered as a part of each recommended activity pattern provide credible information concerning success factors guiding collaboration. Knowledge concerning these factors leads to selection of service entities that are able to execute VO collaborative process instances more efficiently.

The main achievements of this dissertation are the following:

- Identification and evaluation of existing partner and service selection methods in the area of collaborative networked organizations and service-oriented architecture in terms of application to VO collaborative process instantiation (Section 2);
- Identification and evaluation of existing activity recommendation methods in the fields of process-aware information systems, context-aware recommender systems and process mining (Section 3 and 4);
- Formal definition of VO collaborative process, activity pattern, activity pattern context and collaborative process event log (Section 5.4);
- Development of the activity pattern discovery and identification method that permits extraction of activity patterns and their contexts from a collaborative event log maintained by a process-aware information system (Sections 5.5.1 and 5.5.2);
- Development of the method of formulation of recommendations of activity patterns for VO collaborative process executions, where a recommendation is based on the current context of VO collaborative process and activity pattern contexts (Section 5.5.3);
- Development of the activity pattern instantiation method that permits selection of missing actors and service interfaces for activity patterns discovered on the abstract or prototype level; the selection of partners and services is performed within SOVOBE constantly throughout the VO lifecycle (Section 5.5.4);
- Implementation of a prototype of the RMV method composed of the *Operational Support Service, Recommendation Manager, Recommendation Monitor, MatchMaker Module, Process Miner Module, Event log Module* and *Operational Support for Clients Module* (Section 6.1);
- Example integration of the prototype of the RMV method with the process-aware information system named *ErGo* used to support collaboration in the construction sector (Sections 6.2, 6.3 and 6.4);
- Example application of the RMV method to analysis of event log data from a production company, leading to non-trivial, valuable recommendations for Distribution and Warehouse Manager (Section 6.5).

The RMV method is characterized by two important features: extendibility and independence. Its *extendibility* relies on flexible definition of a set of attributes and functions used during activity pattern discovery and recommendation. Different sets of attributes useful in a particular domain or application can be used to describe service entities, service relations, and VO collaborative process instances event contexts. Sophistication of the functions is up to the RMV method user. Such an approach permits both rough and very refined analysis of event logs. The RMV method is *independent* of a particular type of VO collaborative process or type of process-aware information systems. The RMV method can be applied to analysis of any event log that follows the characteristics of collaborative event log. Such independence makes the RMV method applicable to different collaborative process requirements and different business environments. Besides application in the production company (Section 6.5), it is currently under application to analyze document flow processes in Wielkopolska Voivodship Office in Poznań.

Results described in this dissertation were partially presented during "Unleashing Operational Process Mining" Daghstul seminar[15] organized by the IEEE Task Force on Process Mining[16],

[15] "Unleashing Operational Process Mining", Dagstuhl seminar, https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=13481

the World Business Congress organized by the International Management Development Association (Paszkiewicz & Cellary, 2011), two IFIP Working Conferences on Virtual Enterprises (PRO-VE) (Paszkiewicz & Picard, 2010) (Paszkiewicz & Picard, 2009), the 6th International Conference on Theory and Practice of Electronic Governance (Paszkiewicz & Cellary, 2012), the 15th International Conference on Computer Supported Cooperative Work in Design (CSCWD) (Paszkiewicz & Picard, 2011) and two PhD Consortia organized during the Business Information Systems conference (BIS 2013)[17] and East-European Conference in Advances in Databases and Information Systems (ADBIS 2012) conference [18]. The RMV method applications were also published in Journal of Transnational Management (Paszkiewicz & Cellary, 2012) and two book chapters (Picard, et al., 2014) (Picard, et al., 2010). The importance of social network analysis in business process mining was demonstrated in (Paszkiewicz & Picard, 2013)

Discovery and recommendation of activity patterns opens new research directions. Interesting is a possibility of analysis of mutual influence of social networks on process structures and vice versa. This is possible as activity patterns discovered in the RMV method combine two important perspectives on the process: control flow perspective and social perspective. The analysis of mutual impact of interdependent process perspectives on each other is a new and promising research direction that has not yet been explored. Works in this area require extension of the RMV method. Such extension would include simulating and predicting the influence of changes made in one perspective on the structure and characteristics of the other perspective. Analyzed exemplary characteristics of social network perspective include network resilience and integrity. Exemplary characteristics of control flow perspective include process effectiveness or structures of dependencies among activities. Such method would be an important step-ahead in analysis of team dynamics, selection and management of teams and groups of organizations with potential application is various fields, including construction process management and smart cities.

[16] IEEE Task Force on Process Mining, www.win.tue.nl/ieeetfpm/

[17] PhD Symposium co-located with 16th International Conference on Business Information Systems (BIS 2013), http://bis.kie.ue.poznan.pl/16th_bis/phd2013.php

[18] Ph.D. Consortium co-located with 16th East-European Conference in Advances in Databases and Information Systems, http://adbis.cs.put.poznan.pl/call_phd_consortium.php

Bibliography

Aalst, W., 2004. *Discovering Coordination Patterns Using Process Mining.* Bologna, Italy, Springer, pp. 49-63.

Aalst, W., 2006. *Process Mining and Monitoring Processes and Services: Workshop Report.* Dagstuhl, Germany, Dagstuhl: Internationales Begegnungs- und Forschungszentrum für Informatik.

Aalst, W., 2009. *TomTom for Business Process Management (TomTom4BPM).* Amsterdam, The Netherlands, Springer-Verlag, p. 2–5.

Aalst, W., 2011. *Process Mining. Discovery, Conformance and Enhancement of Business Processes.* : Springer.

Aalst, W., 2013. Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior. *IEEE Transactions on Services Computing,* 6(4), pp. 525 - 535.

Aalst, W., De Beer, H. & Van Dongen, B. F., 2005. *Process Mining and Verification of Properties: An Approach based on Temporal Logic.* Agia Napa, Cyprus, Springer-Verlag.

Aalst, W. et al., 2009. *ProM: The Process Mining Toolkit.* Ulm, Germany, CEUR-WS.org.

Aalst, W. M. P., 2004. Business Process Management: A Personal View. *Business Process Management Journal,* II(10).

Aalst, W. M. P., Hofstede, A. H. M. & Weske, M., 2003. *Business Process Management: a Survey.* Berlin/Heidelberg, Springer-Verlag, p. 1–12.

Aalst, W. M. P., Weske, M. & Wirtz, G., 2003. Advanced Topics in Workflow Management: Issues, Requirements, and Solutions. *Journal of Integrated Design & Process Science,* 7(3), p. 49–77.

Aalst, W., Pesic, M. & Schonenberg, H., 2009. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - Research and Development,* II(23), p. 99–113.

Aalst, W., Pesic, M. & Song, M., 2010. Beyond Process Mining - From the Past to Present and Future. *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering,* pp. 38-52.

Aalst, W., Reijers, H. & Song, M., 2005. Discovering Social Networks from Event Logs. *Computer Supported Cooperative Work,* 14(6), pp. 549 - 593.

Aalst, W. et al., 2009. Process Mining: a Two-Step Approach to Balance Between Underfitting and Overfitting. *Software and Systems Modeling,* IX(1), pp. 87-111.

Aalst, W., Schonenberg, M. & Song, M., 2011. Time Prediction Based on Process Mining. *Information Systems,* II(36), p. 450–475.

Aalst, W., Weijters, T. & Maruster, L., 2004. Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering,* 16(9), pp. 1128-1142.

Abowd, G. et al., 1997. Cyberguide: A mobile Context-Aware Tour Guide. *Wireless Networks,* V(3), pp. 421-433.

Abowd, G. et al., 1999. *Towards a Better Understanding of Context and Context-Awareness.* London, UK, Springer-Verlag, pp. 304-307.

Abramowicz, W., 2008. *Filtrowanie informacji.* Poznań: Poznań University of Economics Press.

Abramowicz, W., Haniewicz, K., Kaczmarek, M. & Zyskowski, D., 2008. *E-marketplace for Semantic Web Services.* Berlin/Heidelberg, Springer, p. 271–285.

Adomavicius, G., Sankaranarayanan, R., Sen, S. & Tuzhilin, A., 2005. Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach. *ACM Transactions on Information Systems,* 1(23), pp. 103-145.

Adomavicius, G. & Tuzhilin, A., 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering,* XVII(6), pp. 734-749.

Adomavicius, G. & Tuzhilin, A., 2008. *Context-Aware Recommender Systems.* New York, NY, USA, ACM, pp. 335-336.

Adomavicius, G. & Tuzhilin, A., 2008. *Context-Aware Recommender Systems.* New York, NY, USA, ACM, pp. 335-336.

Agrawal, R. & Srikant, R., 1995. *Mining sequential patterns.* Washington, DC, USA, EEE Computer Society, pp. 3-14.

Akman, V. & Surav, M., 1996. Steps Toward Formalizing Context. *AI Magazine.*

Akman, V. & Surav, M., 1997. The Use of Situation Theory in Context Modeling. *Computational Intelligence,* 13(3), pp. 427-438.

Almeida, T., Vieira, S. & Casanova, M., 2004. *Flexible Workflow Execution Through an Ontology-based Approach.* Vancouver, Canada, Springer.

Anand, S. & Mobasher, B., 2007. Contextual Recommendation. *WebMine,* Issue 4737, pp. 142-160.

Anderson, J., 2009. *Cognitive Psychology and Its Implications.* Seventh Edition ed. : Worth Publishers.

Arazy, O., Kumar, N. & Shapira, B., 2009. Improving Social Recommender Systems. *IT Professional,* IV(11), p. 38–44.

Barba, I., Weber, B. & Del Valle, C., 2012. *Supporting the Optimized Execution of Business Processes Through Recommendations.* Clermont-Ferrand, France, Springer, pp. 135-140.

Barney, J., 1991. Firm Resources and Sustained Competitive Advantage. *Journal of Management,* I(17), p. 99–120.

Barros, A., Dumas, M. & Bruza, P., 2005. *The Move to Web Service Ecosystems.* [On-line] Available at: http://www.bptrends.com/publicationfiles/12-05-WP-WebServiceEcosystems-Barros-Dumas.pdf
[Accessed 19 October 2012].

Bazire, M. & Brézillon, P., 2005. Understanding Context Before Using It. Volume 3554, pp. 113-192.

Berry, M. & Linoff, G., 1997. *Data Minig Techniques: for Marketing, Sales, and Customer Support.* New York, NY, USA: John Wiley & Sons.

Bettini, C. et al., 2010. A Survey of Context Modelling and Reasoning Techniques. *Pervasive and Mobile Computing,* 2(6), pp. 161-180.

Bettman, J., Luce, M. & Payne, J., 1991. *Customer Decision Making: a Consecutive Perspective.,* pp. 1-42.

Bichler, M. & Lin, K., 2006. Service-Oriented Computing. *Computer,* p. 99–101.

Bose, J. & Aalst, W., 2009. *Context-aware Trace Clustering: Towards Improving Process Mining Results.* Sparks, Nevada, USA, SIAM, pp. 401-412.

Brendel, R. & Krawczyk, H., 2008. Application of Social Relation Graphs for Early Detection of Transient Spammers. *WSEAS Transactions on Information Science and Applications,* V(3), pp. 267-276.

Brendel, R. & Krawczyk, H., 2010. *Static and Dynamic Approach of Social Roles Identification Using PISNA and Subgraphs Matching.* Taiyuan, IEEE Computer Society, pp. 557 - 560.

Bridge, D., Goker, M., McGinty, L. & Smyth, B., 2006. Case-Based Recommender Systems. *The Knowledge Engineering review,* III(20), p. 315–320.

Brown, P. & Jones, G. J. F., 2002. *Exploiting Contextual Change in Context Aware Retrieval.* NY, USA, ACM New York, pp. 650-656.

Burke, R., 2002. Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction,* XII(4), pp. 331 - 370.

Burke, R., 2007. Hybrid Web Recommender Systems. In: P. Brusilovsky, A. Kobsa & W. Nejdl, eds. *The Adaptive Web.* Berlin/Heidelberg: Springer , p. 377–408.

Camarinha-Matos, L., Afsarmanesh, H. & Ollus, M., 2008. ECOLEAD and CNO Base Concepts. *Methods and Tools for Collaborative Networked Organizations,* p. 3–32.

Camarinha-Matos, L. et al., 2007. *A Computer-Assisted VO Creation Framework.* Guimarães, Portugal, Springer, pp. 165-178.

Canfora, G., Di Penta, M., Esposito, R. & Villani, M., 2005. *An approach for QoS-aware service composition based on genetic algorithms.* New York, NY, USA, ACM, pp. 1069-1075.

Cantador, I. & Castells, P., 2009. *Semantic Contextualization in a News Recommender System.* , ACM.

Carolis, B., Mazzzotta, I., Novielli, N. & Silvestri, V., 2009. *Using Common Sense in Providing Personalized Recommendations in the Tourist Domain.* New York, NY, USA, ACM.

Cellary, W., 2006. Sieciowe organizacje wirtualne w sektorze małych i średnich przedsiębiorstw. In: P. Adamczewski & J. Stefanowski, eds. *Nowoczesne systemy informatyczne dla małych i średnich przedsiębiorstw.* Poznań, Poland: Wydawnictwa Wyższej Szkoły Bankowej, pp. 13-24.

Cellary, W., 2009. *Networked Virtual Organizations: A Chance for Small and Medium Sized Enterprises on Global Markets.* Nancy (France), Springer, pp. 73-81.

Cellary, W. & Strykowski, S., 2009. *E-government Based on Cloud Computing and Service-Oriented Architecture.* Bogota (Colombia), ACM Press, p. 5–10.

Cena, F. et al., 2006. Integrating Heterogeneous Adaptation Techniques to Build Flexible and Usable Mobile Tourist Guide. *AI Communications,* IV(19), pp. 369-384.

Claro, D., Albers, P. & Hao, J., 2005. *Selecting Web Services for Optimal Composition.* Orlando, USA, Springer.

Crispim, J. & Sousa, J., 2007. *Multiple Criteria Partner Selection in Virtual enterprises.* Guimarães, Portugal, Springer, pp. 197-206.

Davison, B. & Hirsh, H., 1998. *Predicting Sequences of User Actions.* Madison, WI, USA, AAAI Press.

Delias, P. et al., 2013. *Clustering Healthcare Processes with a Robust Approach.* Rome, Italy, EURO-INFORMS.

Demirkan, H. et al., 2008. Service-Oriented Technology and Management: Perspectives on Research and Practice for the Coming Decade. IV(7), p. 356–376.

Dey, A., 2001. Understanding and Using Context. *Personal and Ubiquitous Computing,* 5(1), pp. 4-7.

Ding, H., Benyoucef, L. & Xie, X., 2003. *A Simulation-Optimization Approach Using Genetic Search for Supplier Selection.* New Orleans, Louisiana, USA, IEEE Computer Society.

Dorn, C., Burkhart, T., Werth, D. & Dustdar, S., 2010. *Self-Adjusting Recommendations for People-driven Ad-hoc Processes.* Hoboken, NJ, USA, Springer-Verlag Berlin, Heidelberg, pp. 327-342.

Dourish, P., 2004. What We Talk About When We Talk About Context. *Personal and ubiquitous computing,* 1(8), pp. 19-30.

Do, V., Halatchev, M. & Neumann, D., 2000. *A Context Based Approach to Support Virtual Enterprises.* Washington, DC, USA, IEEE Computer Society.

Drozdowski, L. et al., 2005. A Cooperative Model for Implementing Complex Virtual Enterprises. *Foundations of Computing and Decision Sciences,* pp. 39-48.

Dumas, M., Aalst, W. & Hofstede, A. H., 2005. *Process-Aware Information Systems: Bridging People and Software Through Process Technology.* Hoboken, NJ, USA: John Wiley & Sons, Inc..

Ermilova, E. & Afsarmanesh, H., 2007. *Competency and Profiling Management in Virtual Organization Breeding Environments.* Helsinki, Finland, Springer, pp. 131-142.

Ermilova, E. & Afsarmanesh, H., 2010. Competency Modeling Targeted on Boosting Configuration of Virtual Organizations. *Production Planning and Control. The Management of Operations,* II(21), pp. 103-118.

Fisher, G., 2001. User Modeling in Human-Computer Interaction. *User Modeling and User-Adapted Interaction,* XI(1-2), p. 65–86.

Free Dictionary, 2013. [On-line]
Available at: http://www.thefreedictionary.com/context
[Accessed 26 April 2013].

Gallon, M., Stillman, H. & Coates, D., 1995. Putting Core Competency Thinking Into Practice. *Research Technology Management,* III(38), pp. 20-29.

Ghattas, J., Peleg, M., Soffer, P. & Denekamp, Y., 2009. *Learning the Context of a Clinical Process.* Ulm, Germany, Springer Berlin Heidelberg, pp. 545-556.

Golbeckm, J., 2006. *Generating Predictive Movie Recommendations from Trust in Social Networks.* Pisa, Italy, Springer-Verlag, pp. 93-104.

Gonga, R. et al., 2009. Context Modeling and Measuring for Proactive Resource Recommendation in Business Collaboration. *Journal Computers and Industrial Engineering,* LVII(1), pp. 27-36.

Greenberg, S., 2001. Context as a Dynamic Construct. *Human-Computer Interaction,* XVI(2), pp. 257-268.

Groh, G. & Ehmig, C., 2007. *Recommendations in Taste Related Domains: Collaborative Filtering vs. Social Filtering.* New York, NY, USA, ACM, p. 127–136.

Gunther, C. & Aalst, W., 2007. *Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics.* Berlin, Germany, Springer-Verlag, pp. 328-343.

Haisjackl, C. & Weber, B., 2011. *User Assistance During Process Execution - an Experimental Evaluation of Recommendation Strategies.* Hoboken, New Jersey, USA, Springer, pp. 135-145.

Hartmann, M. & Schreiber, D., 2007. *Prediction Algorithms for User Actions.* Silicon Vally, USA, IEEE Computer Society, pp. 49-354.

Heierman, E. & Cook, D., 2003. *Improving Home Automation by Discovering Regularly Occurring Device Usage Patterns.* Melbourne, Florida, USA, IEEE Computer Society, pp. 537 - 540.

Herlocker, J. & Konstan, J., 2001. Content-Independent Task-Focused Recommendation. *IEEE Internet Computing,* pp. 40-47.

Herlocker, J., Konstan, J., Terveen, L. & Riedl, J., 2004. Evaluating Collaborative Filtering Recommender Systems. *ACM Trans. Information Systems,* XXII(1), pp. 5-53.

Hornix, P., 2007. *Performance Analysis of Business Processes Through Process Mining,* Eindhoven: Technische Universiteit Eindhoven.

Hwang, S., Wei, C. & Yang, W., 2004. Discovery of Temporal Patterns from Process Instances. *Computers in Industry - Special issue: Process/Workflow mining,* 53(3), pp. 345 - 364.

IBM & SAP, 2005. *WS-BPEL Extension for Sub-Processes.* [On-line] Available at: http://pds10.egloos.com/pds/200808/14/94/BPEL-SPE.pdf [Accessed 30 Match 2011].

Jaeger, M. & Mühl, G., 2007. *QoS-based Selection of Services: The Implementation of a Genetic Algorithm.* Bern, Switzland, IEEE Computer Society.

Jannach, D., Zanker, M., Felfernig, A. & Friedrich, G., 2010. *Recommender Systems: An Introduction.* Cambridge: Cambridge University Press.

Jarimo, T. S. A., 2009. Multicriteria Partner Selection in Virtual Organizations With Transportation Costs and Other Network Interdependencies. *IEEE Transactions on Systems, Man and Cybernetics — part C: Applications and reviews,* I(39), pp. 124-129.

Jeong, H., Néda, Z. & Barabási, A.-L., 2003. Measuring Preferential Attachment for Evolving Networks. *Europhysics Letters,* LXI(4), pp. 567-572.

Jones, G. J. F., Glasnevin, D. & Gareth, I., 2005. Challenges and Opportunities of Context-Aware Information Access. *International Workshop on Ubiquitous Data Management,* pp. 53-62.

Koller, D. & M, S., 1996. Toward Optimal Feature Selection. *Proceedings of the 13th International Conference on Machine Learning,* pp. 284-292.

Krawczyk, H. & Brendel, R., 2006. *Spam Classification Methods Based on Users e-mail Communication Graphs.* İstanbul, Turcja, Kadir Has Universitesi, pp. 219-229.

Leoni, M. & Aalst, W., 2013. *Aligning Event Logs and Process Models for Multi-Perspective Conformance Checking: An Approach Based on Integer Linear Programming.* Beijing, China, Springer-Verlag, pp. 113-129.

Liu, H. & Motoda, H., 1998. *Feature Selection for Knowledge Discovery and Data Mining.* Kluwer Academic Publishers ed. Norwell, MA, USA: Springer.

Lombardi, S., Anand, S. & Gorgolione, M., 2009. *Context and Customer Behavior in Recommendation.* New York, NY, USA, ACM.

Magnusson, M., 2004. Repeated Patterns in Behavior and Other Biological Phenomena. *Evolution of Communication Systems : A Comparative Approach,* pp. 111-128.

Magnusson, M., 2005. Understanding Social Interaction: Discovering Hidden Structure With Model and Algorithms. Volume VII, pp. 3-22.

Mahmood, T. & Ricci, F., 2007. *Towards Learning User-Adaptive State Models in a Conversational Recommender System.* Halle, Germany, , pp. 373-378.

Mane, R., 2013. A comparative study of Spam and PrefixSpan sequential pattern mining algorithm for protein sequences. *Advances in Computing, Communication, and Control Communications in Computer and Information Science*, Volume 361, pp. 147-155.

McCarthy, J. & Buvac, S., 1994. *Formalizing Context (Expanded notes),* Stanford: Stanford University.

Merriam-Webster On-line, 2013. [On-line]
Available at: http://www.merriam-webster.com/dictionary/context
[Accessed 26 April 2013].

Ministry of Administration and Digitization, 2012. *Electronic Platform of Public Administration Services (ePUAP).* [On-line]
Available at: http://epuap.gov.pl/wps/portal/
[Accessed 19 October 2012].

Mitchell, M., 1998. *An Introduction to Genetic Algorithms.* Cambridge, MA, USA: MIT Press.

Montali, M. et al., 2010. Declarative Specification and Verification of Service Choreographies. *ACM Transactions on the Web,* I(4), p. 1–62.

Morzy, M., 2012. *An Analysis of Communities in Different Types of Online Forums.* Odense, IEEE Computer Society, pp. 341 - 345.

Morzy, M., 2013. Evolution of Online Forum Communities. In: T. Özyer, J. Rokne, G. Wagner & A. Reuser, eds. *The Influence of Technology on Social Network Analysis and Mining.* Germany: Springer, pp. 615-630.

Morzy, M. & Forenc, K., 2013. Social Network Analysis on Highly Aggregated Data: What Can We Find?. In: T. Morzy, T. Härder & R. Wrembel, eds. *Advances in Databases and Information Systems.* Germany: Springer, pp. 195-206.

Morzy, M., Wierzbicki, A. & and Papadopoulos, A. N., 2009. Mining Online Auction Social Networks for Reputation and Recommendation. *Control and Cybernetics,* XXXVIII(1), pp. 87-106.

Mueller, E., 2006. Production planning and operation in competence-cell based networks. *Production Planning and Control,* II(17), pp. 99-112.

Nakatumba, J., Westergaard, M. & Aalst, W., 2012. *A Meta-model for Operational Support. BPM Center Report.* [On-line]
Available at: http://bpmcenter.org/wp-content/uploads/reports/2012/BPM-12-05.pdf
[Accessed 26 May 2012].

Nardi, B. A., 1995. Studying Context: A Comparison of Activity Theory, Situated Action Models, and Distributed Cognition. *Context and consciousness.*

OASIS Technical Committee, 2006. *Reference Model for Service Oriented Architecture 1.0. OASIS Standard..* [On-line]
Available at: https://www.oasis-open.org/committees/download.php/19679/
[Accessed 17 5 2013].

OASIS, 2007. *WS-Coordination Standard Specification.* [On-line]
Available at: http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.1-spec-errata-os/wstx-wscoor-1.1-spec-errata-os.html
[Accessed 19 October 2012].

Oferia.pl, 2012. [On-line]
[Accessed 19 October 2012].

Oku, K., Nakajima, S., Miyazaki, J. & Uemura, S., 2006. *Context-Aware SVM for Context-Dependent Information Recommendation.* Nara, Japan, IEEE Computer Society, pp. 109-113.

Padovitz, A., Loke, S., Zaslavsky, A. & Burg, B., 2004. *Towards a General Approach for Reasoning About Context, Situations and Uncertainty in Ubiquitous Sensing: Putting Geometrical Intuitions to Work.* Tokyo, Japan, Springer.

Palmisano, C., Tuzhilin, A. & Goegogline, M., 2008. Using the Context to Improve Predicting Modeling of Customers in Personalization Applications. *IEEE Transitions on Knowledge and Data Enginering,* 11(20), pp. 1535-1549.

Panniello, U., Tuzhilin, A., Gorgoglione, M. & Palmisano, C., 2009. *Experimental Comparison Pre- vs. Post-filtering Approaches in Context-Aware Recommender Systems.* New York, NY, USA, ACM, pp. 265-268.

Panorama Firm, 2012. [On-line]
Available at: http://panoramafirm.pl/
[Accessed 19 October 2012].

Pariser, E., 2011. *Beware Online "Filter Bubbles"*. [On-line]
Available at: http://www.youtube.com/watch?v=B8ofWFx525s
[Accessed 18 March 2013].

Pastor-Satorras, R. & Vespignani, A., 2001. Epidemic Spreading in Scale-Free Networks. *Physical Review Letters.*

Paszkiewicz, Z. & Cellary, W., 2011. *Computer supported collaborative processes in virtual organizations.* Poznań, IMDA Press, pp. 85-94.

Paszkiewicz, Z. & Cellary, W., 2012. Computer supported collaboration of SMEs in transnational market. *Journal of Transnational Management,* 17(4), pp. 294-313.

Paszkiewicz, Z. & Cellary, W., 2012. Computer supported collaboration of SMEs in transnational market. *Journal of Transnational Management,* 17(4), pp. 294-313.

Paszkiewicz, Z. & Cellary, W., 2012. *Computer supported contractor selection for public administration ventures.* Albany, NY, ACM, pp. 322-335.

Paszkiewicz, Z. et al., 2011. *ErGo: Developer's Guide,* Poznań: Poznań University of Economics, Department of Information Technology.

Paszkiewicz, Z., Krysztofiak, K., K., W. & Gabryszak, P., 2012. *ErGo: User's Guide,* Poznań: Poznań University of Economics, Department of Information Technology.

Paszkiewicz, Z. & Picard, W., 2009. *Modeling virtual organization architecture with the Virtual Organization Breeding Methodology.* Thessaloniki, Greece, Springer, pp. 187-196.

Paszkiewicz, Z. & Picard, W., 2010. *MAPSS, a Multi-Aspect Partner and Service Selection method.* Saint-Etienne, France, Springer, pp. 329-337.

Paszkiewicz, Z. & Picard, W., 2011. *Modeling competences in service-oriented virtual organization breeding environments.* Lausanne, Switzerland, IEEE, pp. 497-502.

Paszkiewicz, Z. & Picard, W., 2013. *Analysis of the Volvo IT Incident and Problem Handling Processes using Process Mining and Social Network Analysis.* Beijing, China, CEUR online proceedings.

Pazzani, M. & Billsus, D., 2007. Content-Based Recommendation Systems. In: P. Brusilovsky, A. Kobsa & W. Nejdl, eds. *The Adaptive Web.* Berlin/Heidelberg: Springer , pp. 325-341.

Pei, J. et al., 2004. Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering,* Volume 16.

Pepiot, G., Cheikhrouhou, N., Furbringer, J. & Glardon, R., 2007. UECML: Unified Enterprise Competence Modelling Language. *Computers in Industry,* Issue 58, p. 130–142.

Pešic, M., Schonenberg, H. & Aalst, W. M. P., 2007. *DECLARE: Full Support for Loosely-Structured Processes.* Washington, DC, USA, IEEE Computer Society, p. 287–298.

Petrie, C. & Bussler, C., 2008. The Myth of Open Web Services: the Rise of the Service Parks. *IEEE Internet Computing,* III(12), p. 86–95.

Picard, W., 2009. *Social Protocols for Agile Virtual Teams.* Thessaloniki, Greece, Springer, p. 168–176.

Picard, W., 2013. A Formalization of Social Requirements for Human Interactions with Service Protocols. *Information Sciences,* Volume 283, pp. 1-21.

Picard, W., 2013. *Adaptation of Service Protocols.* Poznań, Poland: Poznań University of Economics Press.

Picard, W. & Cellary, W., 2010. *Agile and Pro-active Public Administration as a Collaborative Networked Organization.* New York (NY, USA), ACM, pp. 9-14.

Picard, W. et al., 2010. Breeding Virtual Organizations in a Service-Oriented Architecture Environment. In: *SOA Infrastructure Tools - Concepts and Methods.* Poznań, Poland: Poznań University of Economics Press, p. 375–396.

Picard, W. et al., 2014. Application of the Service-Oriented Architecture at the Inter-Organizational Level. In: S. Ambroszkiewicz, et al. eds. *Studies in Computational Intelligence.* Berlin Heidelberg: Springer, pp. 125-201.

Popescul, A., Ungar, L., Pennock, D. & Lawrence, S., 2001. *Probabilistic Models for Unified Collaborative and Content-Based Recommendation in Sparse-Data Environments.* Seattle, Washington, USA, Morgan Kaufmann, p. 437–444.

Porter, M., 1979. How Competitive Forces Shape Strategy. *Harvard Business Review,* II(57).

Porter, M., 2008. *Competitive Advantage: Creating and Sustaining Superior Performance.* New York, USA: Simon and Schuster.

Rabelo, R. & Gusmeroli, S., 2008. *The ECOLEAD Collaborative Business Infrastructure for Networked Organizations.* Poznań, Springer, p. 451–462.

Ramakrishnan, R. & Gehrke, J., 2000. *Data Management Systems.* : McGraw Hill Companies.

Reichert, M., 2011. *What BPM Technology Can Do for Healthcare Process Support.* Bled, Slovenia, Springer-Verlag, p. 2–13.

Reichert, M., Rinderle, S., Kreher, U. & Dadam, P., 2005. *Adaptive Process Management with ADEPT2.* Washington, DC, USA, IEEE Computer Society, p. 1113–1114.

Ricardo, D., 1817. On the Principles of Political Economy and Taxation.

Ricci, F., Rokach, L., Sshapira, B. & Kantor, P. B., 2011. *Recommender Systems Handbook.* : Springer.

Rozinat, A., 2013. *How to Understand the Variants in Your Process.* [On-line] Available at: http://fluxicon.com/blog/2012/11/how-to-understand-the-variants-in-your-process/
[Accessed 11 January 2013].

Rozinat, A. & Aalst, W., 2006. *Decision Mining in ProM.* Ulm, Germany, Springer, pp. 420-425.

Rozinat, A. & Aalst, W., 2008. Conformance Checking of Processes Based on Monitoring Real Behavior. *Information Systems,* 33(1), pp. 64-95.

Rozinat, A. et al., 2009. Workflow Simulation for Operational Decision Support. *Data and Knowledge Engineering,* IX(68), pp. 834-850.

Russell, N. & Aalst, W., 2007. *Evaluation of the BPEL4People and WS-HumanTask Extensions to WS-BPEL 2.0 Using the Workflow Resource Patterns,* Eindhoven, The Netherlands: Department of Technology Management, Eindhoven University of Technology.

Sadiq, S., Orlowska, M. & Sadiq, W., 2005. Specification and Validation of Process Constraints for Flexible Workflows. *Journal of Information Systems,* 30(5).

Sarwar, B., Karypis, G., Konstan, J. & Riedl, J., 2001. *Item-Based Collaborative Filtering Recommendation Algorithms.* New York, NY, USA, ACM , pp. 285-295 .

Schafer, J., Frankowski, D., Herlocker, J. & Sen, S., 2007. Collaborative Filtering Recommender Systems. In: P. Brusilovsky, A. Kobsa & W. Nejdl, eds. *The Adaptive Web.* Berlin/Heidelberg: Springer, p. 291–324.

Schilit, B. & Theimer, M. M., 1994. Disseminating Active Map Information to Mobile Hosts. *IEEE network,* 5(8), pp. 22-32.

Schilt, B., Adams, N. & Want, R., 1994. Context-Aware Computing Applications. *IEEE Network,* Volume V, pp. 22-32.

Schonenberg, H., Weber, B., Dongen, B. & Aalst, W., 2008. *Supporting Flexible Processes Through Recommendations Based on History.* Milan, Italy, Springer-Verlag, Berlin, pp. 51-66.

Sigg, S., 2008. *Development of Novel Context Prediction Algorithm, and Analysis of Context Prediction Schemes.* Kassel, Germany, Kassel University Press.

Simon, D. & Boring, J., 1990. Sensitivity, Specificity, and Predictive Value. In: H. Walker, W. Hall & J. Hurst, eds. *Clinical Methods: The History, Physical, and Laboratory Examinations.* Boston: Butterworths, p. Chapter 6.

Sinur, J. & Jones, T., 2012. *Leverage Automated Business Process Discovery for Business Benefits,* Stamford, CT, USA: Gartner Report.

Skopik, F., Schall, D., Dustdar, S. & Sesana, M., 2010. *Context-Aware Interaction Models in Cross-Organizational Processes.* Barcelona, IEEE Computer Society, pp. 85 - 90.

Song, M. & Aalst, W., 2008. Towards Comprehensive Support for Organizational Mining. *Decision Support Systems,* XLVI(1), pp. 300-317.

Spohrer, J. & Maglio, P., 2008. The Emergence of Service Science: Toward Systematic Service Innovations to Accelerate Co-creation of Value. *Production and Operations Management,* III(17), p. 238–246.

Srikant, R. & Agrawal, R., 1996. *Mining sequential patterns: generalizations and performance improvements.* London, UK, Springer-Verlag.

Staffware, 2003. *Staffware Process Suite Version 2 – White Paper,* Alkmaar, The Netherlands: Staware PLC.

Stefanidis, K., Pitoura, E. & Vassiliadis, P., 2007. A Context-Aware Preference Database System. *International Journal of Pervasive Computing and Communication,* 4(3), pp. 439-600.

Stoitsev, T., Scheidl, S. & Spahn, M., 2007. *A Framework for Light-weight Composition and Management of Ad-hoc Business Processes.* Toulouse, France, Springer, p. 213–226.

Stoner, J., Freeman, R. & Gilbert, D. R., 1999. *Management.* 6 ed. Singapore: Pearson.

Suneetha, K. & Krishnamoorti, R., 2010. *Advanced Version of Apriori Algorithm.* Washington, DC, USA, IEEE Computer Society, pp. 238-245 .

Swenson, K., 2010. *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done.* Tampa, USA: Meghan-Kiffer Press.

Swinkels, G., 2012. *Performance Improvement based on Cross-Organizational Recommendations,* Eindhoven : Eindhoven University of Technology.

Świerzowicz, J. & Picard, W., 2009. *Social Requirements for Virtual Organization Breeding Environments.* Thessaloniki, Greece, Springer, p. 614–622.

Taghipour, N., Kardan, A. & Ghidary, S., 2007. *Usage-Based Web Recommendations: a Reinforcement Learning Approach.* Minneapolis, MN, USA, ACM, p. 113–120.

Tan, P., Goh, A. & Lee, S., 2008. *A Context Model for B2B Collaborations.* Washington, DC, USA, IEEE Computer Society, pp. 108-115.

U.S. Bureau of Labor Statistics, 2010. *Business Employment Dynamics: Entrepreneurship and the U.S. economy.* [On-line]
Available at: http://www.bls.gov/bdm/entrepreneurship/entrepreneurship.htm.
[Accessed 2012 10 19].

Van Setten, M., Pokraev, S. & Koolwaaij, J., 2004. Context-Aware Recommendations in the Mobile Tourist Application COMPASS. *Adaptive Hypermedia and Adaptive Web-Based Systems,* Volume 3137, pp. 235-244.

Vanderfeesten, I., Reijers, H. & Aalst, W., 2008. *Product Based Workflow Support: Dynamic Workflow Execution.* Montpellier, France, Springer-Verlag, p. 571 – 574.

W3C, 2004. *WS-Choreography Standard Specification.* [On-line]
Available at: http://www.w3.org/TR/ws-chor-model/
[Accessed 19 October 2012].

Wall, Q., 2007. *Rethinking SOA Governance.* [On-line]
Available at: http://www.oracle.com/technetwork/articles/entarch/soagovernance-

093602.html
[Accessed 19 10 2010].

Wang, J. & Han, J., 2004. *BIDE: efficient mining of frequent closed sequences.* Boston, MA, USA, IEEE Computer Society, pp. 79 - 90.

Watts, D., 2004. *Six Degrees: the Science of a Connected Age.* New York, NY, USA: W. W. Norton & Company.

Weber, B., Wild, W. & Breu, R., 2004. Advances in Case-Based Reasoning. *Advances in Case-Based Reasoning,* Volume 3155, pp. 434--448.

Weijters, A. & Aalst, W., 2001. *Process Mining: Discovering Workflow Models from Event-Based Data.* Amsterdam, The Netherlands, Springer-Verlag, pp. 283-290.

Wiszniewski, B., 2011. *Inteligentne wydobywanie informacji z internetowych serwisów.* Gdańsk, Polska: Pomorskie Wydawnictwo Naukowo-Techniczne.

Witten, I., Frank, E. & Hall, M., 2011. *Data Mining. Practical Machine Learning Tools and Techniques.* Third ed. Burlington, MA, USA: Elsevier Inc..

Workflow Management Coalition, 1999. *Terminology and Glossary.* [On-line] Available at: http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf [Accessed 19 October 2012].

YAWL Foundation, 2012. *YAWL.* [On-line]
Available at: http://www.yawlfoundation.org/
[Accessed 20 June 2013].

Yu, Z. et al., 2006. Supporting Context-Aware Media Recommendations for Smart Phones. *IEEE Pervasive Computing,* III(5), pp. 68-75.

List of Figures

List of Tables

List of Listings

Appendix A. Service Protocols Formal Model

In this appendix, the concept of service protocols is formally defined. The formal model of service protocols was first introduced by Picard (Picard, 2013). Examples used in this section have also been introduced in (Picard, 2013).

Definition A.1. (Service Entity) A *service entity* is a common name for an actor or a service interface.

Definition A.2. (Service Entity Description) A *service entity description se* is an object describing a service entity.

Let $SE^* = \{se\}$ denote the set of all the service entity descriptions.

Definition A.3. (Service Network) A *service network* $\sigma = \langle SE, L \rangle$ is a directed graph of service entity descriptions.

Example A.1. Service networks aim at capturing attributes and relations among service entity descriptions, i.e., service consumer descriptions, service interface descriptions, and service provider descriptions. A service network presented in Fig. A.1 consists of $DevHouse$, $ArchibaldTex$, $MoniBank$ represented by rectangles. Service entity descriptions are connected by arcs represented by arrows. For example service entity descriptions $ArchibaldTex$ and $DevHouse$ are connected by an arc modeling the fact of present or past *Collaboration* between the architect and the real-estate developer. Objects describing the arcs are represented by rectangles stuck to the arrows. The attributes of both the service entity descriptions and the objects describing the arcs are represented by inner rounded rectangles.

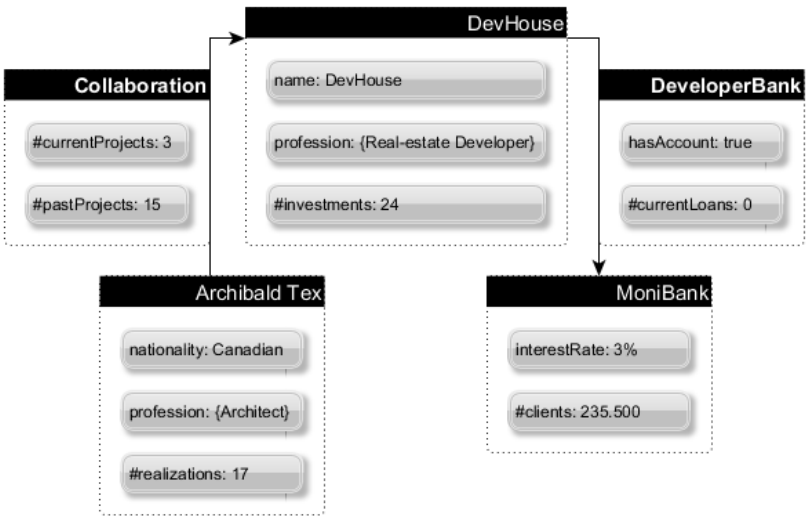


Fig. A.1. Service network example

Definition A.4. (Service Description) A *service description* is a triple defining the "who" (the service consumer), "what" (the service interface), and "whose" (the service provider) part of the activity.

Formally, a *service description* is a triple $s_d = \langle sc^\alpha, si^\alpha, sp^\alpha \rangle \in S_d$, where sc^α is a class of service consumer description, si^α is a class of service interface description, and sp^α is a class of service provider description.

Example A.2. The $ExpiriencedArchitect^\alpha$ is a class of service consumer description. Following this example, a class of service interface description defining access to printing services is defined with constraints such as:

- $\langle \text{CADPlottingSupport}, \supset \{bond, vellum\} \rangle$, and
- $\langle \text{PaymentMeans}, \supset \{BankTransfer, CreditCard\} \rangle$.

Similarly, an example of a class of service provider description defines construction printing companies with constraints concerning the industry sector, the geographical location, etc.

Definition A.5. (Service Requirement) A *service requirements* is a set of attribute constraints that the arcs of service network being instances of that class must observe.

Let l^α denote a service requirement.

Example A.3. Consider a $Collaboration^\alpha$ service requirement, defined by the following set of attribute constraints:

- $\langle \text{No. currentProjects:} > 2 \rangle$,
- $\langle \text{No. pastProjects}, > 5 \rangle$.

The use of *Collaboration*$^\alpha$ service requirements is presented in Fig. A.2. $ExpiriencedArchitect^\alpha$ and $ExpiriencedDeveloper^\alpha$ nodes are connected by an arc associated with the $Collaboration^\alpha$ class.

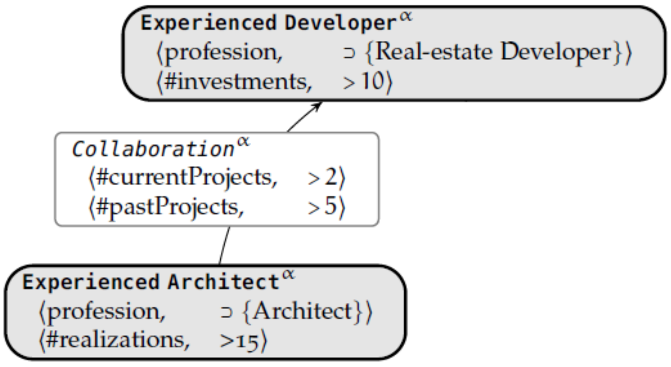

Experienced Developer$^\alpha$
⟨profession, ⊃ {Real-estate Developer}⟩
⟨#investments, > 10⟩
Collaboration$^\alpha$
⟨#currentProjects, > 2⟩
⟨#pastProjects, > 5⟩
Experienced Architect$^\alpha$
⟨profession, ⊃ {Architect}⟩
⟨#realizations, >15⟩

Fig. A.2. Service requirement and classes of objects

Definition A.6. (Service-Oriented Summary of a Process Model) A *service-oriented summary of a process model* is an association of each activity description with a service description.

Formally, a *service-oriented summary of a process model* π_{sos} is a triple $\langle \pi^\alpha, S_d, \rho \rangle$, where π^α is a process model, S_d is a set of service descriptions, and $\rho^\alpha: VD \to S_d$ is a function mapping activity descriptions in π^α to service descriptions in a bijective manner, i.e., $\forall s_d \in S_d, \exists!\, vd \in VD$, such that $\rho(vd) = s_d$.

Let SC^α denote the set of all the classes of service consumer descriptions defined in the service-oriented summary of a process model π_{sos}. Let SI^α denote the set of all the classes of service interface descriptions defined in the service-oriented summary of a process model

π_{sos}. Let SP^{α} denote the set of all the classes of service provider descriptions defined in the service-oriented summary of a process model π_{sos}. Let $S*^{\alpha} = SC^{\alpha} \cup SI^{\alpha} \cup SP^{\alpha}$.

Example A.4. The concept of service-oriented summary of a process model is presented in Fig. A.3, in which three service descriptions are represented as rectangles on the left side. Each service description contains a class of service consumer descriptions, represented by a rounded rectangle labeled $sc\{i\}$, a class of service interface descriptions, represented by a rounded rectangle labeled $si\{i\}$, and a class of service provider descriptions, represented by a rounded rectangle labeled $sp\{i\}$, where $i \in \{1,2,3\}$. Next, three dashed arrows are connecting service descriptions with the activity descriptions of the process model represented by rounded rectangles labeled *a1*, *a2*, and *a3*. These three arrows visualize the mapping function ρ. Rounded rectangles labeled *e1*, *e2*, and *e3* represent arcs connecting activities.

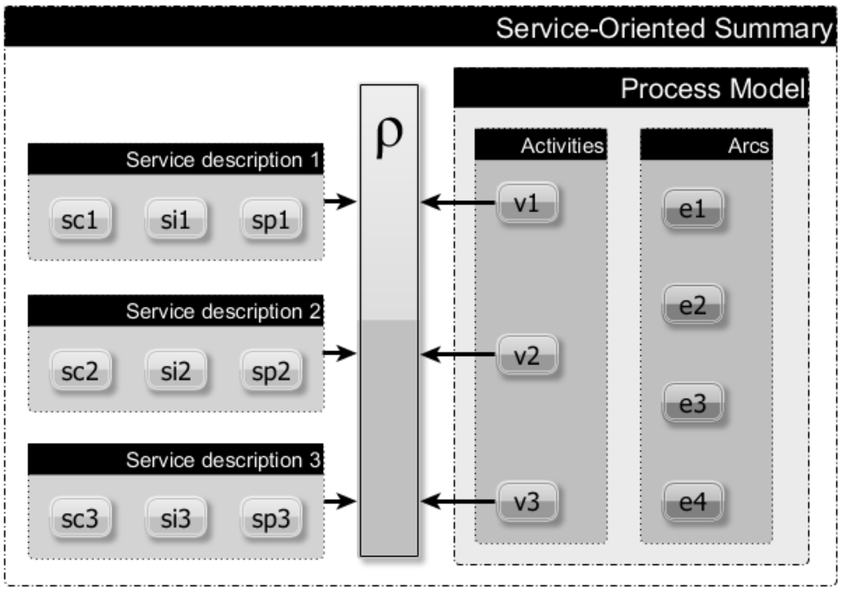


Fig. A.3. Service-oriented summary of a process model

Definition A.7. (Service Network Schema) A *service network schema* is a graph $\sigma^{\alpha} = \langle SE^{\alpha}, L^{\alpha} \rangle$ composed of classes of service entity descriptions $SE^{\alpha}$and service requirements L^{α}.

Example A.5. Service network schema aims at capturing classes of service entity descriptions and their relations. A service network schema presented in Fig. A.4 consists of $ExpiriencedDeveloper^{\alpha}$, $ExpiriencedArchitect^{\alpha}$ and $Bank^{\alpha}$ classes represented by rectangles. Classes of service entity descriptions are connected by service requirements represented by arrows. For example, classes of service entity descriptions $ExpiriencedDeveloper^{\alpha}$ and $ExpiriencedArchitect^{\alpha}$ are connected by a service requirement $Collaboration^{\alpha}$ modeling constraints on possible collaboration between the architect and the real-estate developer. Service requirements are represented by rectangles stuck to the arrows. The attribute constraints of both classes of service entity descriptions and service requirements are represented by inner rounded rectangles.

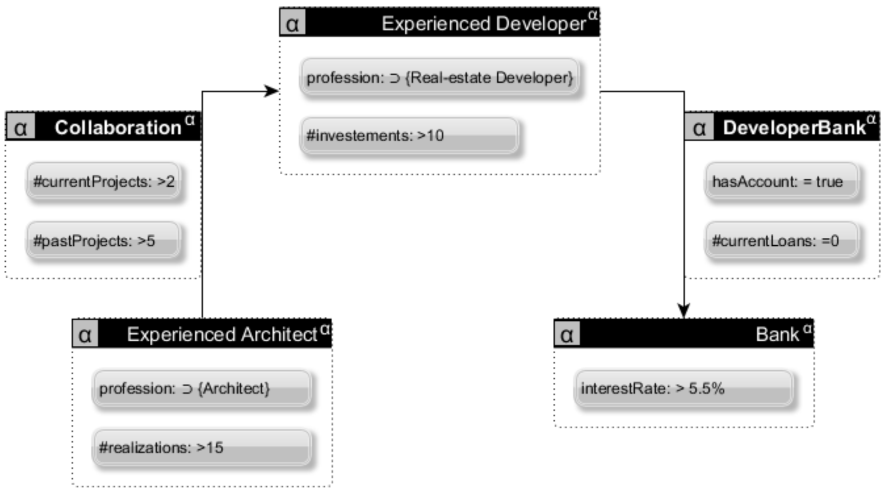


Fig. A.4. Service network schema

Definition A.8. (Abstract Service Protocol) A *service protocol* Y^{α} is a triple $Y^{\alpha} = \langle \pi^{\alpha}_{sos}, \sigma^{\alpha}, \overset{\alpha}{\Lambda} \rangle$ where π^{α}_{sos} is a service-oriented summary of a process model π^{α}, σ^{α} is a graph representing a service network schema, and $\overset{\alpha}{\Lambda}$ is a mapping relations between the service-oriented summary of a process model and service network schema.

The mapping relation $\overset{\alpha}{\Lambda} : (S *^{\alpha} = SC^{\alpha} \cup SI^{\alpha} \cup SP^{\alpha}) \times SE^{\alpha}$ associates elements of service descriptions – service customer description, service interface description, service provider classes description – with classes of service entity descriptions of the service network schema. Each element of a service description $s *^{\alpha}$ – sc^{α}, $si^{\alpha}$and sp^{α} – is associated with the class of service entity description se^{α} of the service network schema. Formally, $\forall (s *^{\alpha}, se) \in S *^{\alpha} \times SE^{\alpha}, s *^{\alpha} \overset{\alpha}{\Lambda} se^{\alpha}$

Example A.6. An abstract service protocol is presented in Fig. A.5. The service-oriented summary of the abstract service protocol is represented at the top of the Fig. A.5. The service network schema is represented at the bottom of the Fig. A.5. A set of dashed arrows associates the elements of service descriptions of the service-oriented summary with the nodes of the service network schema represented by rounded rectangles labeled $v\{i\}$, where $i \in [1,9]$. This set of arrows represents the mapping relation $\overset{\alpha}{\Lambda}$ between the service-oriented summary and the service network schema. Therefore, the service consumer description *sc1* is associated with the class of service entity descriptions *v6* of the service network schema. Additionally, the following relations of service description are defined: $sc\langle consumes, = true\rangle si$, $sp\langle provides, = true\rangle si$, $si\langle isConsumedBy, = true\rangle sc$, $si\langle isConsumedBy, = true\rangle sp$. Although implicit in service descriptions, these relations are not explicitly defined in the service network schema.

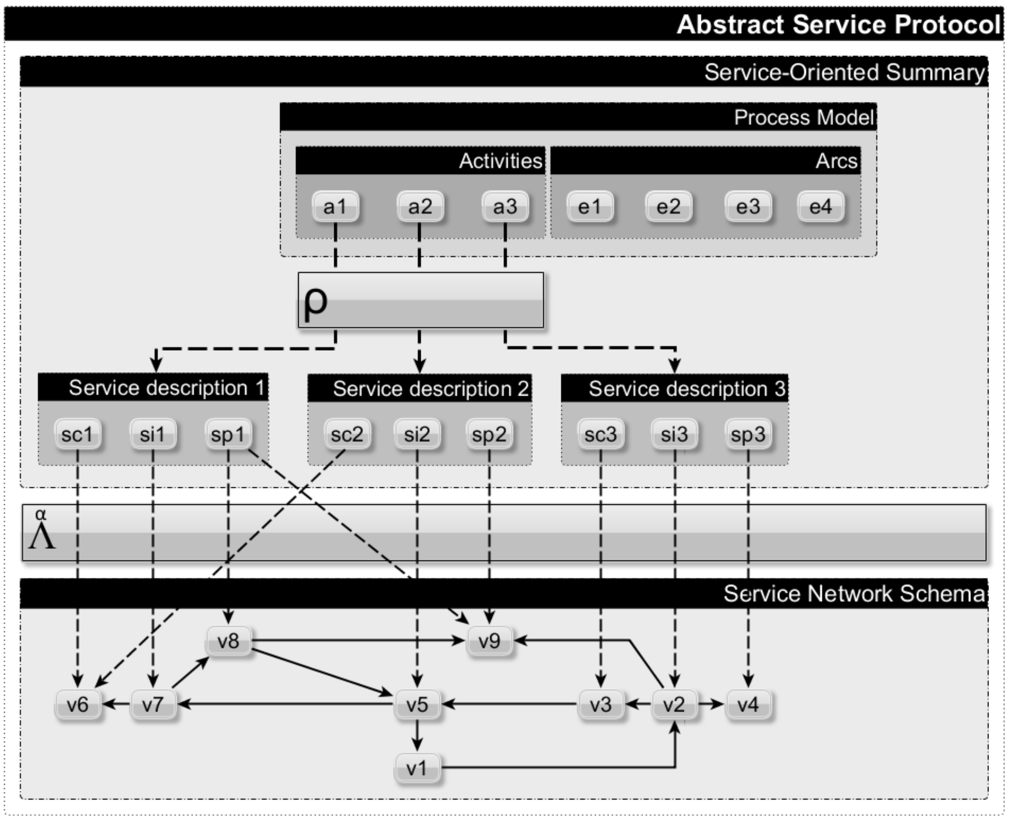


Fig. A.5. Abstract service protocol

Compliance of a service network with a service network schema has a global character, although it is based on the local concept of membership. *Membership* refers to a particular type of relations that may exist between objects and classes in a service network and a service network schema. Although various types of membership may be defined in service networks and service network schemata, class relational membership and link class full membership have to be defined for a further definition of compliance.

Definition A.9. (Class Relational Membership) A service entity description se is a *relational member* of a class of service entity description se^{α}, denoted $se{\subset^{\bullet}}se^{\alpha}$, iff: (1) se is an instance of se^{α}, (2) for each service requirement starting from se^{α} and associated with class *c*, at least one link starting from se is associated with an instance of class *c*, (3) for each service requirement leading to se^{α} and associated with a class *c*, at least one link leading to se is associated with an instance of class *c*.

Definition A.10. (Link Class Full Membership) A link $l = \langle se_{src}, se_{dst}, o\rangle$ is a *full member* of the class of links $l^{\alpha} = \langle se^{\alpha}_{src}, se^{\alpha}_{dst}, c\rangle$ denoted $l^{\bullet}{\subset^{\bullet}}l^{\alpha}$, iff the source and destination service entity descriptions are instances of their respective classes of service entity descriptions, and the object associated with the link is an instance of the class associated with the class of links.

Formally, $l^{\bullet}{\subset^{\bullet}}l^{\alpha} \Leftrightarrow se_{src} \propto se^{\alpha}_{src}$ and $se_{dst} \propto se^{\alpha}_{dst}\ and\ o \propto c.$

Based on the membership relations, the concept of compliance with a service network schema may be defined. A service network is compliant with a service network schema if the constraints on the service entity descriptions and the social requirements among them, defined in a service network schema, are satisfied by a given service network.

Definition A.11. (Compliance Relation) Having a service network schema $\sigma^{\alpha} = \langle SE^{\alpha}, L^{\alpha}\rangle$and service network $\sigma = \langle SE, L\rangle$, a compliance relation $\oplus$ on $\sigma \times \sigma^{\alpha}$ is a relation that: (1) the compliance of a service entity description se with a class of service entity descriptions se^{α} implies that that the service entity description se is a relational member of the class of service entity descriptions se^{α}, (2) for each service requirement l^{α} between two classes of service entity descriptions se^{α}_{src} and se^{α}_{dst}, for each service entity descriptions se_{src} and se_{dst} being members of classes $se^{\alpha}_{src}$and se^{α}_{dst} respectively, there exist a link between that is a full member of l, (3) for each class of service entity descriptions, at least one service entity description is compliant with the class.

Definition A.12. (Partial Compliance Relation) Having a service network schema $\sigma^{\alpha} = \langle SE^{\alpha}, L^{\alpha}\rangle$ and service network $\sigma = \langle SE, L\rangle$, a *partial compliance relation* $\pm$ on $\sigma \times \sigma^{\alpha}$ is a relation that satisfies only condition 1 and 2 from Definition A.11, the third condition is relaxed.

Definition A.13. (Compliance with Service Network Schema) A service network $\sigma = \langle SE, L\rangle$ is *compliant with a service network schema* $\sigma^{\alpha} = \langle SE^{\alpha}, L^{\alpha}\rangle$, denoted $\sigma \oplus \sigma^{\alpha}$, iff there exist a compliance relation $\oplus$ on $\sigma \times \sigma^{\alpha}$.

Definition A.14. (Partial Compliance with Service Network Schema) A service network $\sigma = \langle SE, L\rangle$ is *partially compliant with a service network schema* $\sigma^{\alpha} = \langle SE^{\alpha}, L^{\alpha}\rangle$, denoted $\sigma \pm \sigma^{\alpha}$, iff there exist a partial compliance relation $\pm$ on $\sigma \times \sigma^{\alpha}$.

Example A.7. The service network schema in Fig. A.4 is compliant with service network presented in Fig. A.1. The compliance relation $\oplus$ applies to the following pairs of service entity descriptions and classes:

- $ArchibaldText \oplus ExpiriencedArchitect^{\alpha}$
- $DevHouse \oplus ExpiriencedDeveloper^{\alpha} MoniBank \oplus Bank^{\alpha}$

Additionally, for each service requirement between two classes of service entity descriptions among $ExpiriencedArchitect^{\alpha}$, $ExpiriencedDeveloper^{\alpha}$, and $Bank^{\alpha}$, there is a full member link between two service entity descriptions that are instances of the given classes. For example, the link *Collaboration* is a full member of the service requirement $Collaboration^{\alpha}$. As a consequence, both conditions (1) and (2) from Definition A.11 are satisfied. Also condition (3) is satisfied, as all the classes of service entity descriptions in service network schema have their instances in the service network presented in Fig. A.1.

Definition A.15. (Prototype Service Protocol) A *prototype service protocol* Y^{β} is a tuple $Y^{\beta} = \langle Y^{\alpha}, \sigma, \Omega, \Phi\rangle$ where Y^{β} is an abstract service protocol, σ is a service network, and $\Omega: SE \times S*^{\alpha}$ is a relation associating service entity descriptions of the service network to elements of service descriptions, and $\Phi: SE \times SE^{\alpha}$ is a relations associating nodes of service network to classes of service entity descriptions of the service network schema of Y^{α}.

Example A.8. An example of a prototype service protocol is presented in Fig. A.6. Each service entity description of the service network is represented by a circle, some service entity descriptions being additionally labeled, e.g., the top-right service entity description is labeled *'1'*. The links between service entity descriptions are represented by solid lines between the nodes. The relation Ω mapping service entity descriptions of the service network to elements of service descriptions is represented by dashed arrows on the left. Note that many service entity descriptions may be associated with a given service description element. In Fig. A.6., both nodes labeled *'1'* and *'2'* are associated with the service interface description *si1*. Additionally, the relation Φ mapping nodes of the service network to classes of service entity descriptions of the service network schema is represented by dot-dashed arrows on the right side. Note that many service entity descriptions may be associated with a given class of service entity description of the service network schema. In Fig. A.6, both nodes labeled *'1'* and *'2'* are associated with the class of service entity descriptions *v7*.

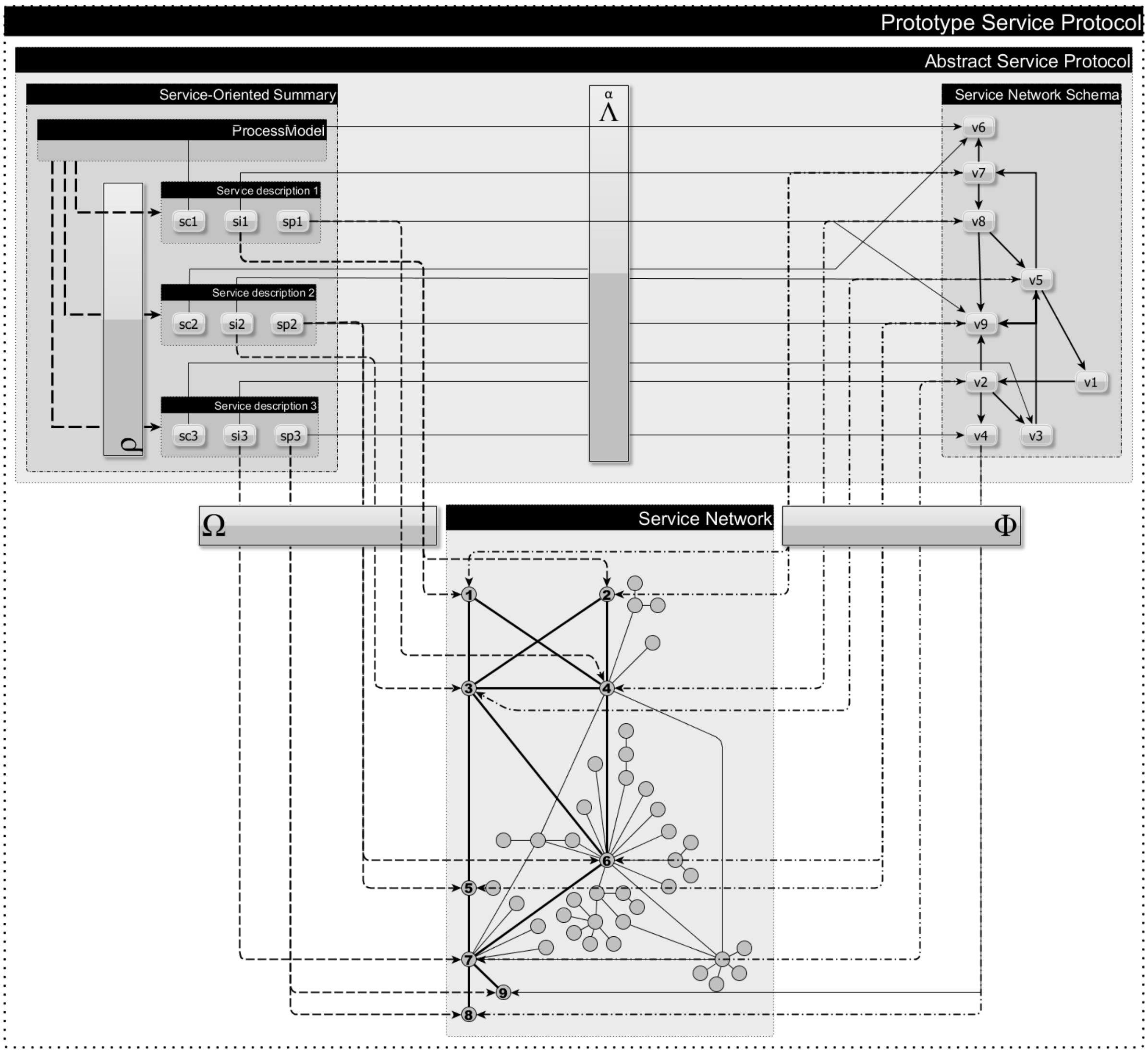


Fig. A.6. Prototype service protocol

When a prototype service protocol is fully implemented, i.e., there is a service entity description implementing each element of service description, then it may be executed. The service protocol is then an *executable service protocol*.

Definition A.16. (Executable Service Protocol) An *executable service protocol* Y^{ε} is a prototype service protocol such that $\forall s*^{\alpha} \in S*^{\alpha}, \exists se \in SE$ such that $se\Omega s*^{\alpha}$ and the induced relation $\Phi^{+}: SE \times SE^{\alpha,+}$ is in compliance relation.

Appendix B. RMV Method Prototype Modules and Classes

Sequence Pattern Discovery module

All the SPD Java^{TM} classes and interfaces are in the `pl.poznan.pue.dit.zp.spd` package. Classes from this package are associated with context classes from `pl.poznan.pue.dit.zp.context` package. Classes of the SPD module are presented in Fig. B.1.

The SPD module classes:

- Provide access to an event log,
- Transform event traces to activity sequences,
- Identify frequent and closed activity sequence patterns,
- Capture event, activity instance description and activity pattern contexts.

The `EventLog` class provides abstraction for event log management. This class is responsible for accessing traces (`getTraces` method), process instance attributes (`getProcessInstanceAttributes` method) and events (`getParallelEvents` method). The `getParallelEvents` method returns events that were recorded in `delta` time span before the event `e`. The `getProcessInstanceAttributes` and `gatParallelEvents` methods are used to determine a context of an activity instance. Finally, the `importTraces` method of `EventLog` class converts traces into activity sequences modeled by `ActivitySequence` class.

The `ActivitySequence` class is an extension of the `AbstractSequence` class. Abstraction for activity sequence persistence is provided by `ActivitySequenceDao` class. The `EventLog` class is associated with the `Trace` class. Each instance of `Trace` class is composed of events modeled by the `Event` class. Each event object is an aggregation of attributes being objects of `Attribute` class.

The `ActivityInstanceDescription` class represents activity instance descriptions. All the activity instance descriptions are created during import of traces from event log – `importTraces` method of `EventLog` class – and their transformation to activity sequences – `createActivitySequence` method of `ActivitySequenceDao` class. The access to events of an activity instance description is provided by `getEvents` method of `ActivityInstanceDescription` class. The `ActivityInstanceDescription` class has also `getStartingEvent`, `getFinishingEvent` methods for retrieving first and last event recorded for the activity instance. The context of the first event is returned by the `getStartingEventContext` method.

Each object of `ActivitySequence` class holds association to particular trace returned by `getTrace` method. Context of an activity instance description is created by the `createContext` method call. The `createContext` method internal logic encompasses: calling `getStartingEventContext` method of

ActivityInstanceDescription class, calling getParallelEvents and getProcessInstanceAttributes methods of an EventLog object, retrieving proceeding events and service entities by calling getProceedingActors and getProceedingEvents methods of the Trace class.

Each activity sequence object is composed of a set of activity instance description sets represented by ActivityInstanceDescriptionSet class. Each activity instance description set is composed of a number of activity instance descriptions. Four methods of ActivityInstanceDescriptionSet class permit comparison of activity instance description sets: containsAll, contains, equal, exaclyEquals. Those methods are used during sequence pattern discovery.

The sequence pattern discovery algorithm is launch by calling findSequencePatternMethod of the ActivitySequenceDatabase class. This method is an implementation of an algorithm presented in Section 5.5.1. Algorithm is configured by an instance of RMVPatternConfirguration class. Configuration includes a subset of parameters presented in Section 5.6 that are relevant for discovery of sequence patterns.

Resulting sequence patterns are represented by SequencePattern class. Similarly to ActivitySequence class, SequencePattern class is an extension of the AbstractSequence class. Each activity pattern object is an aggregation of objects being instances of the Pair class. Each instance of Pair class links the activity instance description with activity sequences supporting this activity instance description. Additionally isPostfix method gives access to information if the activity instance description was a part of the larger set of activity instance descriptions or not. This information is used when merging pairs into sequence patterns in database projection step of the algorithm.

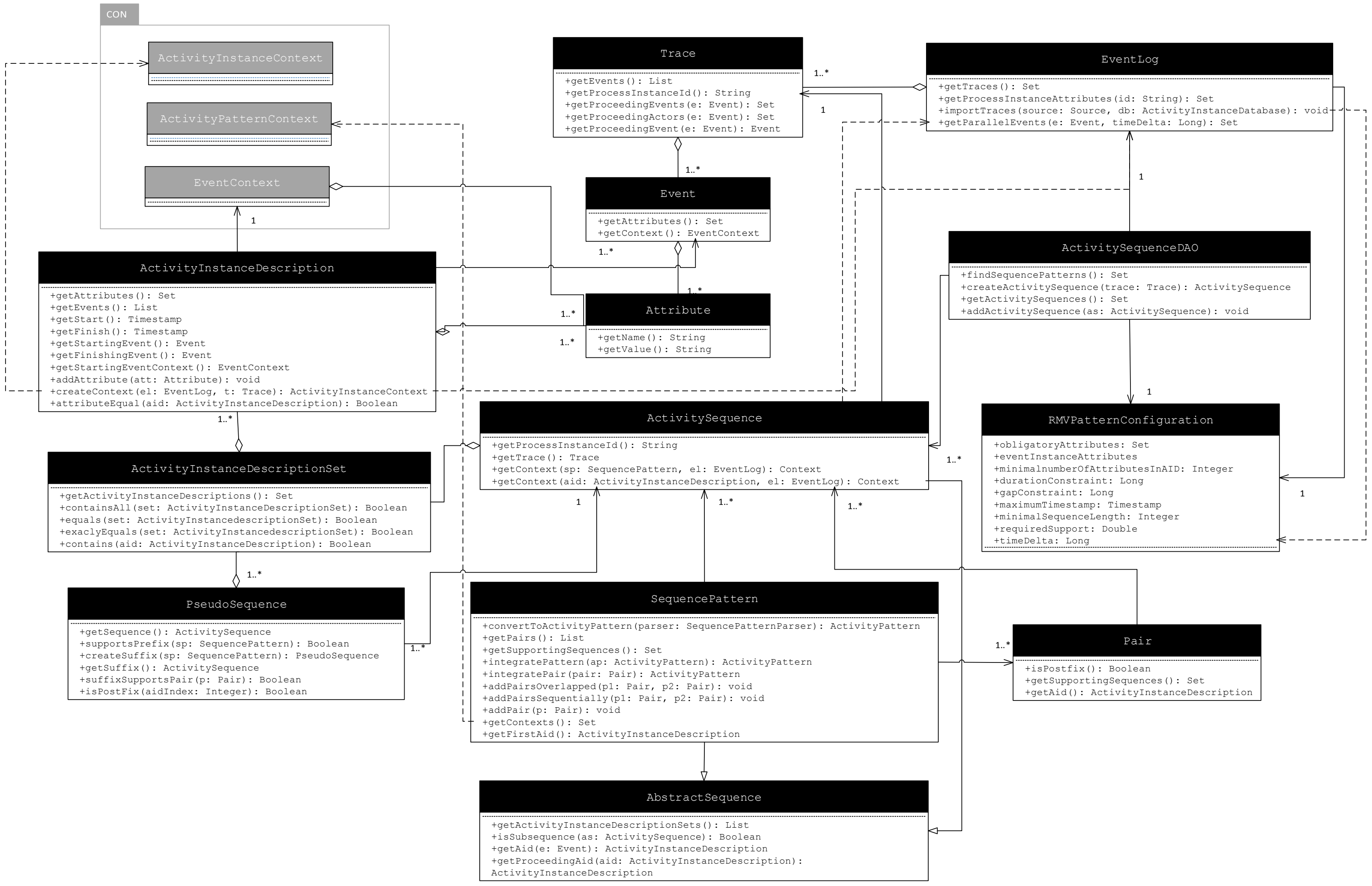

CON
ActivityInstanceContext
ActivityPatternContext
EventContext
Trace
+getEvents(): List
+getProcessInstanceId(): String
+getProceedingEvents(e: Event): Set
+getProceedingActors(e: Event): Set
+getProceedingEvent(e: Event): Event
EventLog
+getTraces(): Set
+getProcessInstanceAttributes(id: String): Set
+importTraces(source: Source, db: ActivityInstanceDatabase): void
+getParallelEvents(e: Event, timeDelta: Long): Set
Event
+getAttributes(): Set
+getContext(): EventContext
Attribute
+getName(): String
+getValue(): String
ActivityInstanceDescription
+getAttributes(): Set
+getEvents(): List
+getStart(): Timestamp
+getFinish(): Timestamp
+getStartingEvent(): Event
+getFinishingEvent(): Event
+getStartingEventContext(): EventContext
+addAttribute(att: Attribute): void
+createContext(el: EventLog, t: Trace): ActivityInstanceContext
+attributeEqual(aid: ActivityInstanceDescription): Boolean
ActivitySequenceDAO
+findSequencePatterns(): Set
+createActivitySequence(trace: Trace): ActivitySequence
+getActivitySequences(): Set
+addActivitySequence(as: ActivitySequence): void
RMVPatternConfiguration
+obligatoryAttributes: Set
+eventInstanceAttributes
+minimalnumberOfAttributesInAID: Integer
+durationConstraint: Long
+gapConstraint: Long
+maximumTimestamp: Timestamp
+minimalSequenceLength: Integer
+requiredSupport: Double
+timeDelta: Long
ActivitySequence
+getProcessInstanceId(): String
+getTrace(): Trace
+getContext(sp: SequencePattern, el: EventLog): Context
+getContext(aid: ActivityInstanceDescription, el: EventLog): Context
ActivityInstanceDescriptionSet
+getActivityInstanceDescriptions(): Set
+containsAll(set: ActivityInstanceDescriptionSet): Boolean
+equals(set: ActivityInstancedescriptionSet): Boolean
+exaclyEquals(set: ActivityInstancedescriptionSet): Boolean
+contains(aid: ActivityInstanceDescription): Boolean
PseudoSequence
+getSequence(): ActivitySequence
+supportsPrefix(sp: SequencePattern): Boolean
+createSuffix(sp: SequencePattern): PseudoSequence
+getSuffix(): ActivitySequence
+suffixSupportsPair(p: Pair): Boolean
+isPostFix(aidIndex: Integer): Boolean
SequencePattern
+convertToActivityPattern(parser: SequencePatternParser): ActivityPattern
+getPairs(): List
+getSupportingSequences(): Set
+integratePattern(ap: ActivityPattern): ActivityPattern
+integratePair(pair: Pair): ActivityPattern
+addPairsOverlapped(p1: Pair, p2: Pair): void
+addPairsSequentially(p1: Pair, p2: Pair): void
+addPair(p: Pair): void
+getContexts(): Set
+getFirstAid(): ActivityInstanceDescription
Pair
+isPostfix(): Boolean
+getSupportingSequences(): Set
+getAid(): ActivityInstanceDescription
AbstractSequence
+getActivityInstanceDescriptionSets(): List
+isSubsequence(as: ActivitySequence): Boolean
+getAid(e: Event): ActivityInstanceDescription
+getProceedingAid(aid: ActivityInstanceDescription):
ActivityInstanceDescription
1..*
1
1..*
1..*
1..*
1..*
1..*
1
1
1
1..*
1..*
1..*
1
1..*
1..*
1..*
1
1..*

Fig. B.1. Classes in Sequence Pattern Discovery module

A context of an sequence pattern is retrieved by calling `getContexts` method. For each activity sequence supporting sequence pattern returned by `getSupporting` method, a set of methods is called to collect: event context, process instance attributes, proceeding actors and activity instance descriptions, parallel events (*cf.* Section 5.4.2). The interaction among objects of various classes during retrieval of sequence pattern context is presented on UML sequence diagram in Fig. B.2.

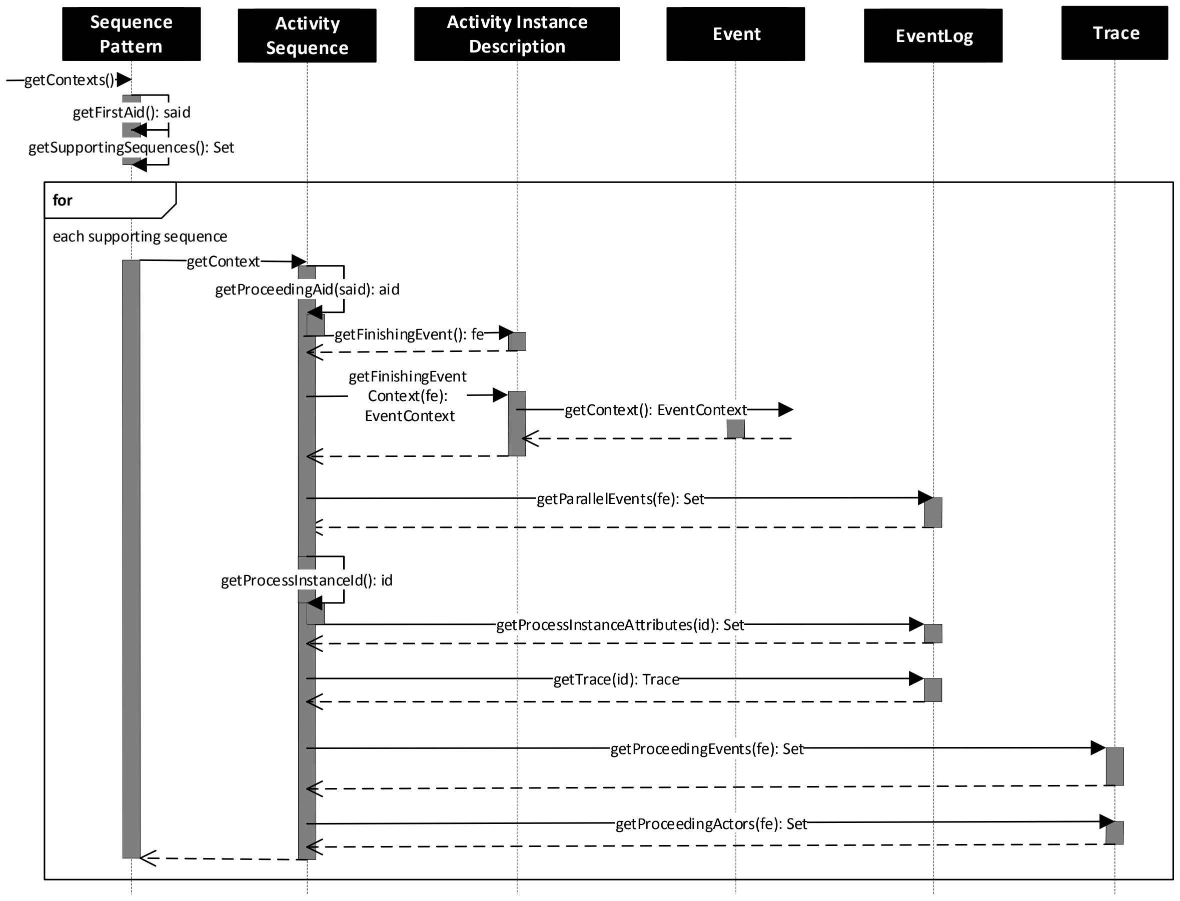

Sequence Pattern
Activity Sequence
Activity Instance Description
Event
EventLog
Trace
getContexts()
getFirstAid(): said
getSupportingSequences(): Set
for
each supporting sequence
getContext
getProceedingAid(said): aid
getFinishingEvent(): fe
getFinishingEvent Context(fe): EventContext
getContext(): EventContext
getParallelEvents(fe): Set
getProcessInstanceId(): id
getProcessInstanceAttributes(id): Set
getTrace(id): Trace
getProceedingEvents(fe): Set
getProceedingActors(fe): Set

Fig. B.2. Requesting sequence pattern contexts

Sequence pattern object can be transformed to activity pattern by calling `convertToActivityPattern` method of the `SequencePattern` class.

The key class for discovery of sequence patterns is the `PseudoSequence` class. Objects of the class are used during the database projection step of the algorithm. The `createSuffix` and `getSuffix` methods provide access to suffix created for: (1) the sequence returned by `getSequence` method, and (2) prefix being the sequence pattern.

The summary of functionality provided by the SPD module classes is presented in Tab. B.1

.

Tab. B.1. Functionality of SPD module classes

Function	Classes
Access to an event log	`EventLog`
Transformation of event traces to activity sequences Capturing context of events, activity instance descriptions and activity patterns	`Traces`, `Event`, `EventLog`, `ActivitySequenceDatabase`, `ActivitySequence`, `AbstarctSequence`, `ActivityInstanceDescription`, `ActivityInstanceDescriptionSet`, `Attribute`, `RMVPatternConfiguration`
Identification of frequent activity sequence patterns	`SequencePattern`, `PseudoSequence`, `Pair`, `ActivitySequenceDatabase`, `RMVPatternConfiguration`

Activity Pattern Identification module

All the SPD classes and interfaces are in the `pl.poznan.pue.dit.zp.api` package and use classes from `pl.poznan.pue.dit.zp.recommendation.context` and `pl.poznan.pue.dit.zp.activitysequences` packages.

Classes from API module:

- Model the concept of activity patterns,
- Transform sequence patterns to activity patterns,
- Store activity patterns and provide access to their context.

The classes presented in Fig. B.3 corresponds to the formal model of activity patterns introduced in Section Activity Pattern Identification and in Appendix A.

The activity pattern is modeled by the `ActivityPattern` class. Each activity pattern object is associated with abstract activity pattern being an instance of `AbstractActivityPattern` class. Classes `ProcessModel`, `ActivityDescription`, `ServiceDescription` correspond to service-oriented summary of a process model of an activity pattern. Service network schema is modeled by the `SericeNetworkSchema` class. This class is associated with `ServiceEntityClass` and `ServiceRequierement` classes corresponding to concepts of class of service entity and service requirement respectively. Similarly, `ServiceNetwork` class is associated with `ServiceEntity` and `ServiceRelation` classes. Classes `Implementer` and `Mapping` correspond to mapping relations among service entities and classes of service entities, and service description and classes of service entities respectively. The `createInstance` method of the `ActivityPattern` class return function that maps service entities to elements of service descriptions. The `ServiceEntity` class has two extensions: `ServiceIntefacedescription` and `ActorDescription` classes. These classes represent service interfaces and actors registered in the RMV prototype, that are assigned to the activity pattern.

Conversion of a sequence pattern to an activity pattern is started by calling the `convertToActivityPattern` method of the `SequencePattern` (*cf.* Fig. B.2). This method uses the object of the `SequencePatternParser` class to perform all the necessary conversions. The `SequencePatternParser` class implements activity pattern identification method presented in Section 5.5.2. Methods `createServiceOrientedSummaryOfProcessModel`, `createServiceNetworkSchema`, `createServieNetwork`, `createMapping` correspond to steps from Listing 5.8. Parsing algorithm uses configuration parameters represented by the object of `RmvParsingConfiguration` class (*cf.* Section 5.6). Created activity patterns are added to activity pattern database – the `ActivityPatternDao` class hides a logic behind storing and retrieving activity patterns.

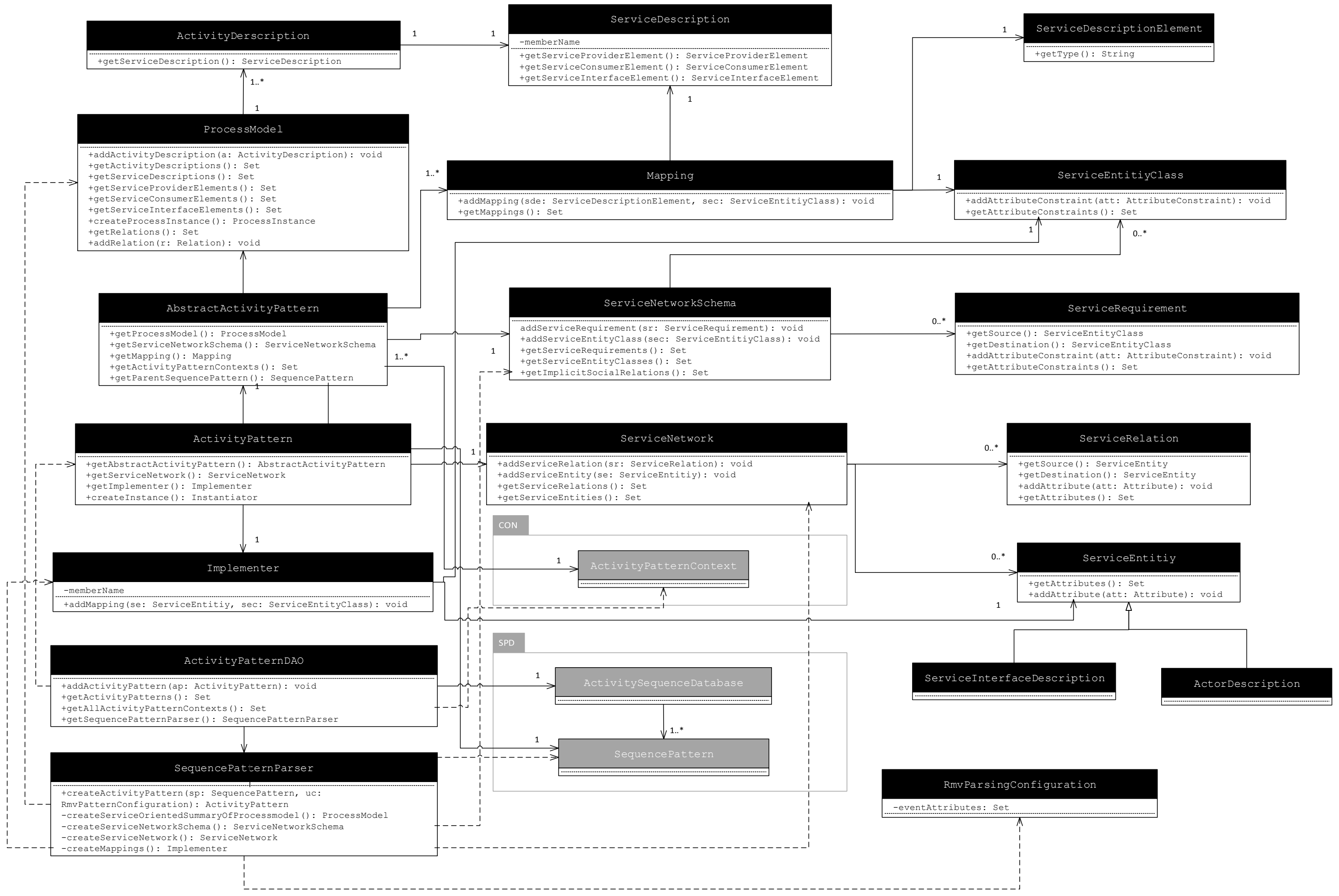

ActivityDerscription
+getServiceDescription(): ServiceDescription
ServiceDescription
-memberName
+getServiceProviderElement(): ServiceProviderElement
+getServiceConsumerElement(): ServiceConsumerElement
+getServiceInterfaceElement(): ServiceInterfaceElement
ServiceDescriptionElement
+getType(): String
ProcessModel
+addActivityDescription(a: ActivityDescription): void
+getActivityDescriptions(): Set
+getServiceDescriptions(): Set
+getServiceProviderElements(): Set
+getServiceConsumerElements(): Set
+getServiceInterfaceElements(): Set
+createProcessInstance(): ProcessInstance
+getRelations(): Set
+addRelation(r: Relation): void
Mapping
+addMapping(sde: ServiceDescriptionElement, sec: ServiceEntitiyClass): void
+getMappings(): Set
ServiceEntitiyClass
+addAttributeConstraint(att: AttributeConstraint): void
+getAttributeConstraints(): Set
AbstractActivityPattern
+getProcessModel(): ProcessModel
+getServiceNetworkSchema(): ServiceNetworkSchema
+getMapping(): Mapping
+getActivityPatternContexts(): Set
+getParentSequencePattern(): SequencePattern
ServiceNetworkSchema
addServiceRequirement(sr: ServiceRequirement): void
+addServiceEntityClass(sec: ServiceEntitiyClass): void
+getServiceRequirements(): Set
+getServiceEntityClasses(): Set
+getImplicitSocialRelations(): Set
ServiceRequirement
+getSource(): ServiceEntityClass
+getDestination(): ServiceEntityClass
+addAttributeConstraint(att: AttributeConstraint): void
+getAttributeConstraints(): Set
ActivityPattern
+getAbstractActivityPattern(): AbstractActivityPattern
+getServiceNetwork(): ServiceNetwork
+getImplementer(): Implementer
+createInstance(): Instantiator
ServiceNetwork
+addServiceRelation(sr: ServiceRelation): void
+addServiceEntity(se: ServiceEntitiy): void
+getServiceRelations(): Set
+getServiceEntities(): Set
ServiceRelation
+getSource(): ServiceEntity
+getDestination(): ServiceEntity
+addAttribute(att: Attribute): void
+getAttributes(): Set
CON
ActivityPatternContext
Implementer
-memberName
+addMapping(se: ServiceEntitiy, sec: ServiceEntityClass): void
ServiceEntitiy
+getAttributes(): Set
+addAttribute(att: Attribute): void
SPD
ActivityPatternDAO
+addActivityPattern(ap: ActivityPattern): void
+getActivityPatterns(): Set
+getAllActivityPatternContexts(): Set
+getSequencePatternParser(): SequencePatternParser
ActivitySequenceDatabase
ServiceInterfaceDescription
ActorDescription
SequencePatternParser
+createActivityPattern(sp: SequencePattern, uc: RmvPatternConfiguration): ActivityPattern
-createServiceOrientedSummaryOfProcessmodel(): ProcessModel
-createServiceNetworkSchema(): ServiceNetworkSchema
-createServiceNetwork(): ServiceNetwork
-createMappings(): Implementer
SequencePattern
RmvParsingConfiguration
-eventAttributes: Set
1
1..*
1..*
0..*
0..*
0..*

Fig. B.3. Classes in Activity Pattern Identification module

The `getAllActivityPatternContext` method returns all the context of all the activity patterns stored in activity pattern database. The contexts of particular activity pattern are retrieved by calling the `getContexts` method of the `AbstractActivityPattern` class. The logic of this method calls the `getContexts` method of parent sequence pattern returned by `getParentsequencePattern` method. Corresponding UML sequence diagram is presented in Fig. B.4.

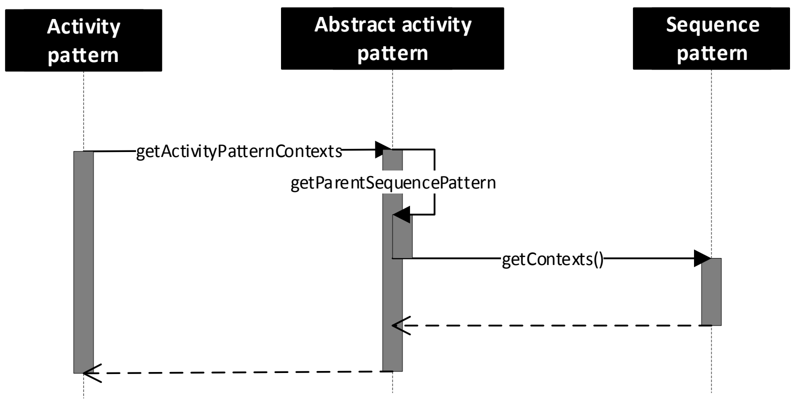


Fig. B.4. Requesting activity pattern contexts

The summary of functionality provided by the API module classes is presented in Tab. B.2.

Tab. B.2. Functionality of API module classes

Function	**Classes**
Activity patterns model	`ActivityPattern, AbstractActivityPattern, Activitydescription, ServiceDescription, Mapping, Implementer, ServiecDescriptionElement, ServiceNetworkSchema, ServiceEntityClass, serviceRequirement, ServiceNetwork, ServiceRelation, ServiceEntity, ServiceInterfaceDescription, ActorDescription`
Transformation sequence patterns to activity patterns	`SequencePatternParser, RmvParsingConfiguration`
Activity patterns persistence and access to context	`ActivityPatternDao`

Recommendation Formulation module

Classes of the RF module (Fig. B.5) are stored in `pl.poznan.pue.dit.zp.rf` package. Classes are associated with other modules (`pl.poznan.pue.dit.zp.rm`, `pl.poznan.pue.dit.zp.api`, `pl.poznan.pue.dit.zp.spd`) and supporting packages (`pl.poznan.pue.dit.zp.functions`, `pl.poznan.pue.dit.zp.context`).

Classes from the RF module:

- Implement formal model from Section 5.4.5, and
- Implement recommendation formulation method presented in Section 5.5.3.

Recommendation request is modelled by `RecommendationRequest` class. The access to the current context of a collaborative process instance is provided by `getContext` method of this class. The `getProcessInstanceId` method returns the identifier of the process instance for which the recommendation is requested.

The `RecommendationQuery` class models an input for recommendation engine. Methods of the `RecommendationQuery` class provide access to data necessary for recommendation engine, i.e., recommendation request (`getRecommendationRequest` method), set of activity patterns stored in the RMV prototype (`getClosedActivityPatterns` method). Recommendation formulation algorithm is implemented in `RecommendationEngine` class. Configuration of the algorithm is modeled as the `RmvRecommConfiguration` class. The recommendation is formulated by the `getRecommendation` method.

The `RecommendationEngine` class internally uses the three methods. The `createClass` method returns a context class created on the basis of the context from the recommendation query and the `contextClassFunction` from the `RmvRecommmConfiguration` object. A set of activity patterns is returned by the `findActivityPatterns` method. Finally, *specificity*, *sensitivity*, *weighed context distance*, *support* and *confidence indicator*, *social coefficient* are calculated within `calculatePartialMetrics`. The method returns the object of the `PartialJustification` class. Not that the *confidence indicator* and *social coefficient* values are retrieved from recommendation monitor modeled by `RecommendationMonitor` class.

The recommendation returned by the recommendation engine is modeled by `Recommendation` class. The `Recommendation` class is an aggregation of instances of `RecommendationElement` class. Both `Recommendation` and `RecommendationElement` classes provide access to identifier of a process instance they were created for – `getProcessInstanceId` method. The process instance identifier is copied from recommendation request. The `getActivityPattern` method of `RecommendationElement` class provides access to activity pattern associated with recommendation element.

The justification of the recommendation element is modeled by `FullJustification` class. The three methods the `FullJustification` on the basis of user preferences. Each `FullJustification` object consist of *recomm index* and *nonrecom index* values, and

PartialJustification object. The getCDiff method returns the difference between *recomm index* and *nonrecom index*.

Information from FullJustification class is used by calculateRecommendationValue method to calculate a total attractiveness of the recommendation element. The getSortedRecommendationElements method orders recommendation elements according to values returned by the calculateRecommendationValue method.

User preferences are modeled as UserConfiguration class. Preferences include definition of ri and nri functions used for calculation of *recomm index* and *nonrecom index* values. Note that while recommendation engine uses parameters from RmvRecommConfiguration class that are common for all the users, the configuration modeled with UserConfiguration is specific for particular user. The interaction among objects of various classes during recommendation generation is presented on UML sequence diagram in Fig. B.5. Two phases are distinguished in Fig. B.6: (1) generation of recommendation elements with partial justification, (2) generation of full justification and ordering of recommendation elements following preferences of the particular user.

The best recommendation element is selected to execution by calling promoteToExecution method of RecommendationElement class. Information concerning the selected activity pattern is forwarded to recommendation monitor (RecommendationMonitor class) for update of social coefficient. Recommendations with recommendation elements and partial justifications are stored in database modeled as RecommendationDao.

The summary of functionality provided by the RF module classes is presented in Tab. B.3.

Tab. B.3. Functionality of RF module classes

Function	Classes
Recommendation request	RecommendationRequest, RecommendationQuery
Recommendation generation	RecommendationEngine, PartialJustification, RmvRecommConfiguration, Recommendation, RecommendationElement, RecommendationDao
Recommendation elements ordering	UserConfiguration, Recommendation, RecommendationElement, FullJustification

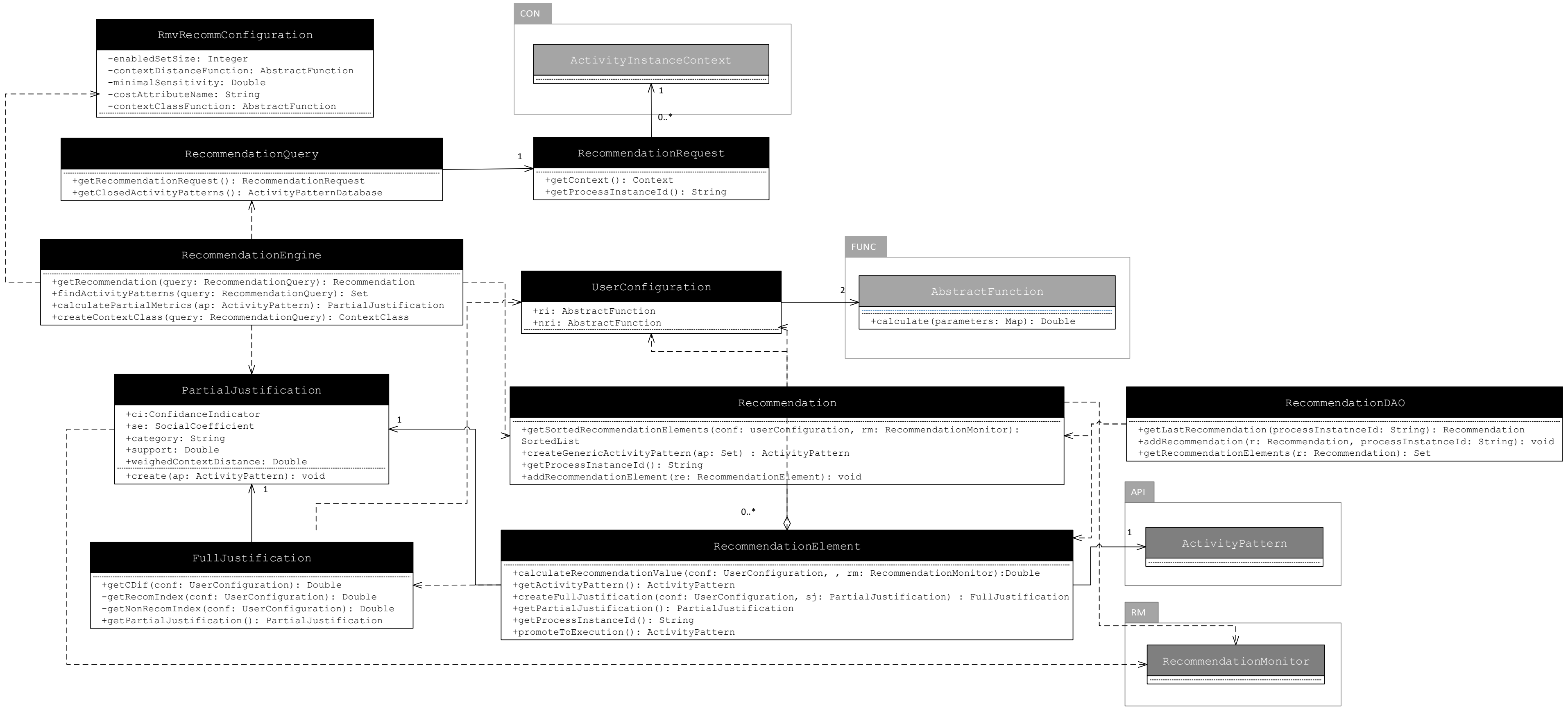

RmvRecommConfiguration
-enabledSetSize: Integer
-contextDistanceFunction: AbstractFunction
-minimalSensitivity: Double
-costAttributeName: String
-contextClassFunction: AbstractFunction
CON
ActivityInstanceContext
1
0..*
RecommendationQuery
+getRecommendationRequest(): RecommendationRequest
+getClosedActivityPatterns(): ActivityPatternDatabase
1
RecommendationRequest
+getContext(): Context
+getProcessInstanceId(): String
RecommendationEngine
+getRecommendation(query: RecommendationQuery): Recommendation
+findActivityPatterns(query: RecommendationQuery): Set
+calculatePartialMetrics(ap: ActivityPattern): PartialJustification
+createContextClass(query: RecommendationQuery): ContextClass
UserConfiguration
+ri: AbstractFunction
+nri: AbstractFunction
2
FUNC
AbstractFunction
+calculate(parameters: Map): Double
PartialJustification
+ci:ConfidanceIndicator
+se: SocialCoefficient
+category: String
+support: Double
+weighedContextDistance: Double
+create(ap: ActivityPattern): void
1
1
Recommendation
+getSortedRecommendationElements(conf: userConfiguration, rm: RecommendationMonitor): SortedList
+createGenericActivityPattern(ap: Set) : ActivityPattern
+getProcessInstanceId(): String
+addRecommendationElement(re: RecommendationElement): void
RecommendationDAO
+getLastRecommendation(processInstatnceId: String): Recommendation
+addRecommendation(r: Recommendation, processInstatnceId: String): void
+getRecommendationElements(r: Recommendation): Set
0..*
API
1
ActivityPattern
FullJustification
+getCDif(conf: UserConfiguration): Double
-getRecomIndex(conf: UserConfiguration): Double
-getNonRecomIndex(conf: UserConfiguration): Double
+getPartialJustification(): PartialJustification
RecommendationElement
+calculateRecommendationValue(conf: UserConfiguration, , rm: RecommendationMonitor):Double
+getActivityPattern(): ActivityPattern
+createFullJustification(conf: UserConfiguration, sj: PartialJustification) : FullJustification
+getPartialJustification(): PartialJustification
+getProcessInstanceId(): String
+promoteToExecution(): ActivityPattern
RM
RecommendationMonitor

Fig. B.5. Recommendation classes in recommendation formulation module

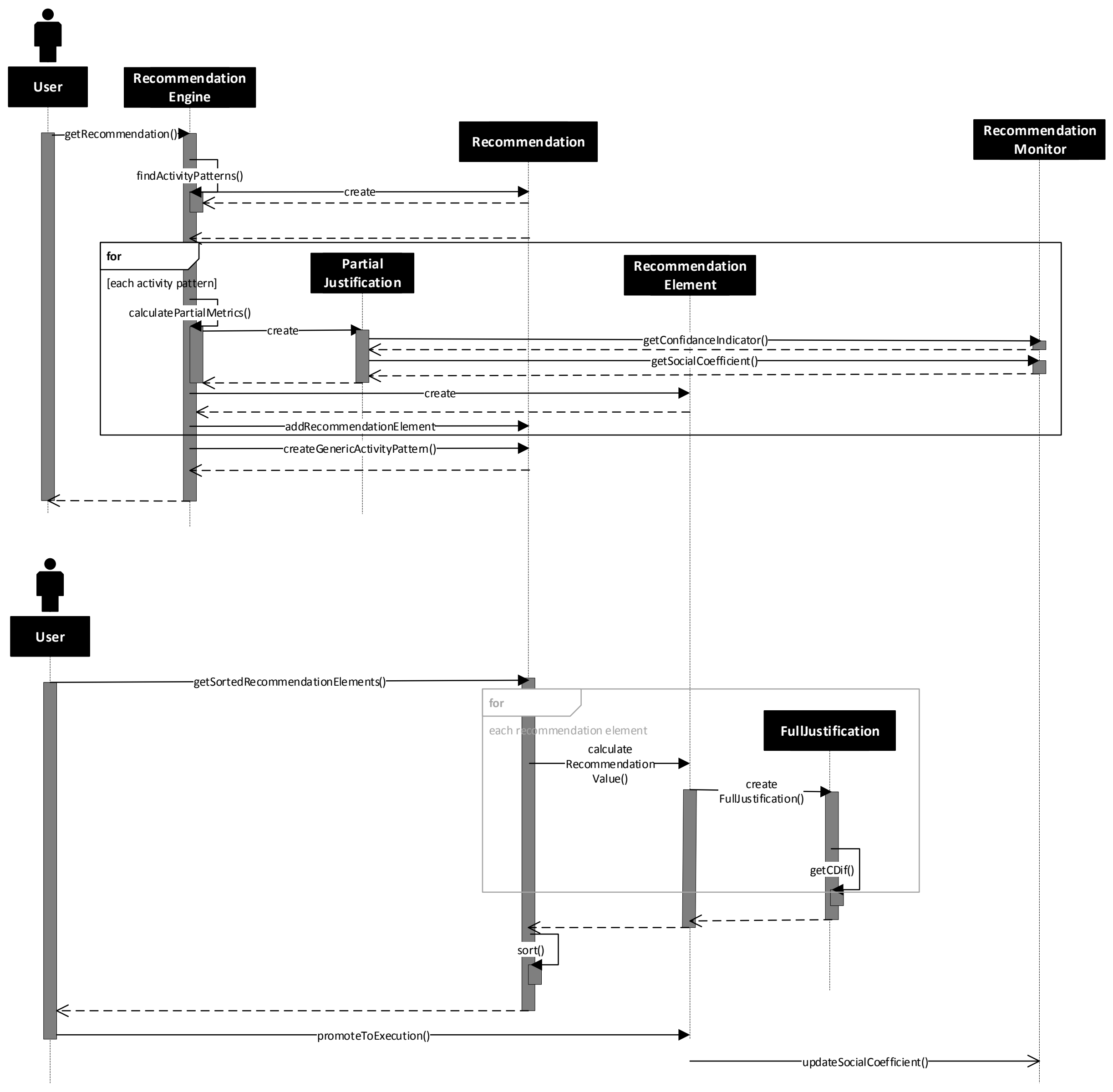


Fig. B.6. Creation of recommendation and sorting recommendation elements according to user preferences

Activity Pattern Instantiation module

Classes stored in `pl.poznan.pue.dit.zp.apin` package are presented in Fig. B.7. Classes from the APIN module:

- Implement formal model from Section 5.4.6, and
- Implement recommendation formulation method presented in Section 5.5.4.

Assignment of service entity to class of service entities is modeled as `ClassVariant` class. The `ClassVariant` class provides methods to access: class of service entity (`getServiceEntityClass` method) and service entity (`getServiceEntity` method). The `evaluate` method returns an evaluation of compliance of service entity to class of service entities based on compliance function provided by a user. The `getActivityPattern` function returns the activity pattern being instantiated. Finally promoteToOffer method performs two operations: (1) creates an object of the ClassOffer class, and (2) saves newly created class offer to database. Objects of the ClassOffer class model assignment of service entities to classes of service entities that is considered especially attractive and worth discussion with other selecting collaborators.

The assignment of class compliance function to particular class of service entity is modeled by ClassFunction class. Each object of the ClassFucntion class has three methods: (1) the getClasscomplianceFunction method is used for evaluation of assignment for particular class of service entity, (2) the getComplianceThreshold methods returns compliance threshold value, and (3) the getServiceEntityClass method returns class of service entity. A set of ClassFucntion objects is modeled by the ClassFunctions models.

Generation of class variants is performed by the generator modeled by the ClassVariantGenerator class. The generateClassVariants method returns a set of service entities assigned to the class of service entity. The generateClassVariants method uses getFunctionByClass method to access user-defined compliances function for the particular class of service entity. A set of service entities generated by the generator for the class of service entity, is a subset of all the services entities registered in the RMV prototype. The getAllClassVariants method returns all the sets of service entities for all the classes of service entities. Access to repository of actors and service interfaces is provided by geActorDescriptionRepository and getServiceInterfaceDescriptionrepository methods. Access to class variants generated by the generator is provided by getSortedVariantsForClass and getGeneratedVariants.

VO variants are molded by the VOVariant class VoVarint is an aggregation of class variants. Each VO variant can be transformed to object of VOOffer class. VOOffer class object is an aggregation of class offers. Note that each VOOffer and ClassOffer classes has evaluate method that takes as an argument a user configuration. This method returns the evaluation of an offer from point of view of a particular user. Persistence of ClassOffer and VOOffer objects is performed by OfferDao class. Both VOOffer and VoVariant class objects can be modified using changeAssignment method. Finally, one VoOffer is selected for instantiation of activity pattern by calling instantiate method of VoOffer class.

VO variants are generated by an instance of VoVariantGenerator class. The VoVariantGenerator class implements the genetic algorithm. Information concerning the best VO variant is returned by the getBestVariant and getBestVariantEvaluation methods.

The summary of functionality provided by the APIN module classes is presented in Tab. B.4.

Tab. B.4. Functionality of APIN module classes

Function	**Classes**
Assignment of service entities to classes of service entities	`ClassFunction`, `ClassFunctions`, `ClassVariantGenerator`, `UserConfiguration`
Evaluation of groups of service entities	`VoVariantGenerator`, `VoVariant`, `UserConfiguration`
Collaboration over selection of service entities	`ClassOffer`, `VoOffer`, `OfferDAO`, `UserConfiguration`

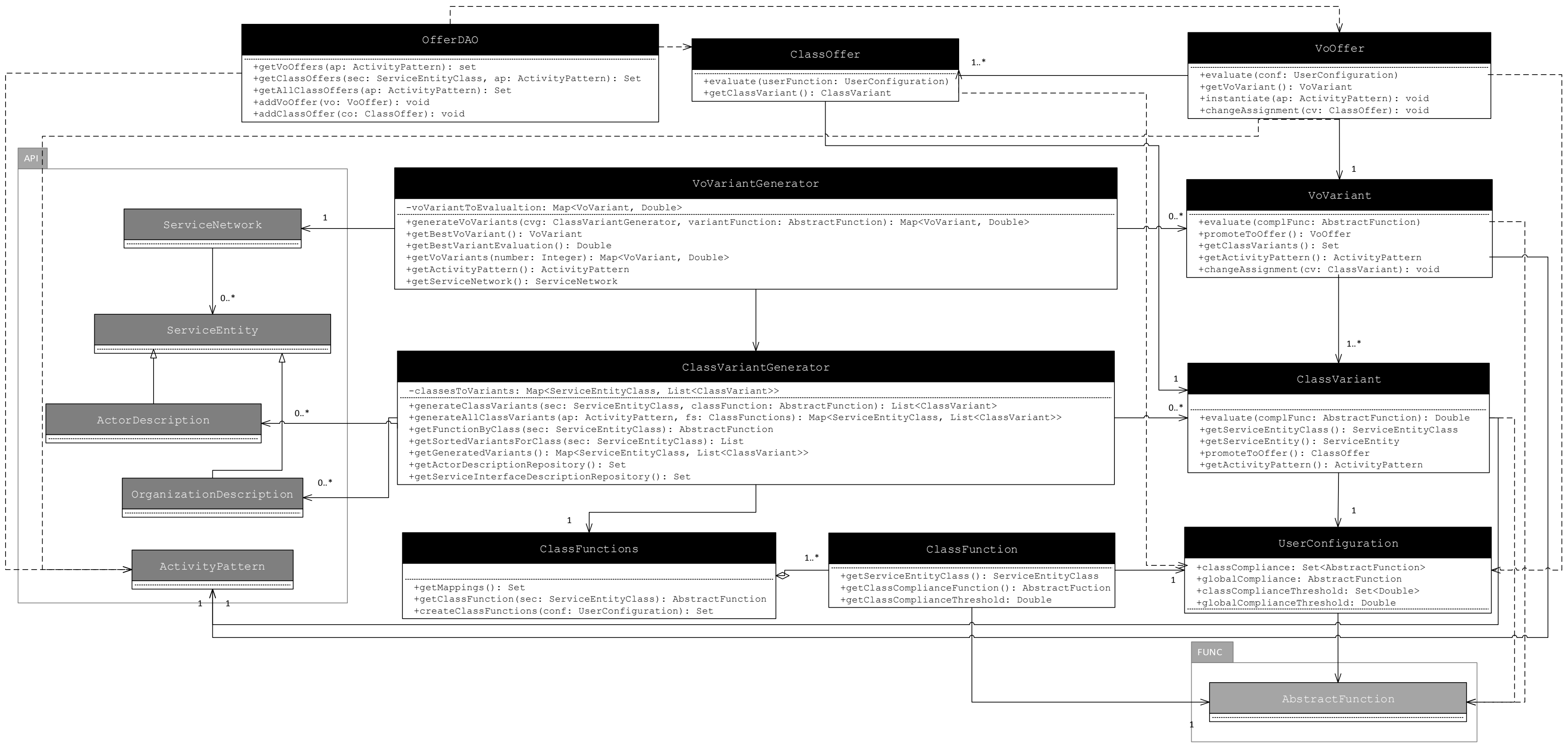

OfferDAO
+getVoOffers(ap: ActivityPattern): set
+getClassOffers(sec: ServiceEntityClass, ap: ActivityPattern): Set
+getAllClassOffers(ap: ActivityPattern): Set
+addVoOffer(vo: VoOffer): void
+addClassOffer(co: ClassOffer): void
ClassOffer
+evaluate(userFunction: UserConfiguration)
+getClassVariant(): ClassVariant
1..*
VoOffer
+evaluate(conf: UserConfiguration)
+getVoVariant(): VoVariant
+instantiate(ap: ActivityPattern): void
+changeAssignment(cv: ClassOffer): void
API
ServiceNetwork
1
VoVariantGenerator
-voVariantToEvaluation: Map<VoVariant, Double>
+generateVoVariants(cvg: ClassVariantGenerator, variantFunction: AbstractFunction): Map<VoVariant, Double>
+getBestVoVariant(): VoVariant
+getBestVariantEvaluation(): Double
+getVoVariants(number: Integer): Map<VoVariant, Double>
+getActivityPattern(): ActivityPattern
+getServiceNetwork(): ServiceNetwork
1
0..*
VoVariant
+evaluate(complFunc: AbstractFunction)
+promoteToOffer(): VoOffer
+getClassVariants(): Set
+getActivityPattern(): ActivityPattern
+changeAssignment(cv: ClassVariant): void
0..*
ServiceEntity
ActorDescription
0..*
ClassVariantGenerator
-classesToVariants: Map<ServiceEntityClass, List<ClassVariant>>
+generateClassVariants(sec: ServiceEntityClass, classFunction: AbstractFunction): List<ClassVariant>
+generateAllClassVariants(ap: ActivityPattern, fs: ClassFunctions): Map<ServiceEntityClass, List<ClassVariant>>
+getFunctionByClass(sec: ServiceEntityClass): AbstractFunction
+getSortedVariantsForClass(sec: ServiceEntityClass): List
+getGeneratedVariants(): Map<ServiceEntityClass, List<ClassVariant>>
+getActorDescriptionRepository(): Set
+getServiceInterfaceDescriptionRepository(): Set
1..*
1
ClassVariant
0..*
+evaluate(complFunc: AbstractFunction): Double
+getServiceEntityClass(): ServiceEntityClass
+getServiceEntity(): ServiceEntity
+promoteToOffer(): ClassOffer
+getActivityPattern(): ActivityPattern
OrganizationDescription
0..*
1
1
ClassFunctions
+getMappings(): Set
+getClassFunction(sec: ServiceEntityClass): AbstractFunction
+createClassFunctions(conf: UserConfiguration): Set
1..*
ClassFunction
+getServiceEntityClass(): ServiceEntityClass
+getClassComplianceFunction(): AbstractFuction
+getClassComplianceThreshold: Double
1
UserConfiguration
+classCompliance: Set<AbstractFunction>
+globalCompliance: AbstractFunction
+classComplianceThreshold: Set<Double>
+globalComplianceThreshold: Double
ActivityPattern
1
1
FUNC
AbstractFunction
1

Fig. B.7. Classes in Activity Pattern Instantiation module

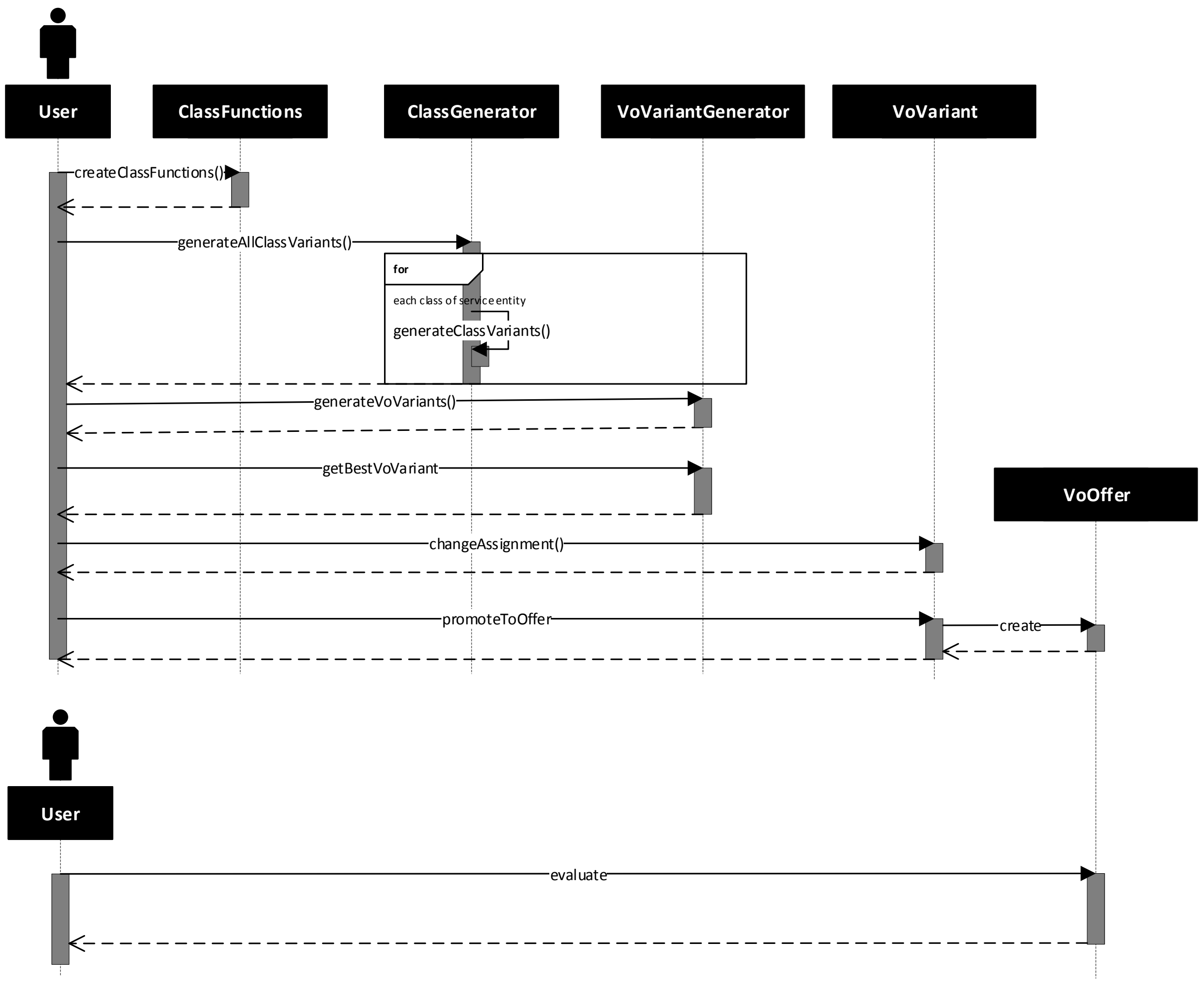


Fig. B.8. Generation, selection and modification of the best Vo Variant

Recommendation Monitoring module

Classes stored in the `pl.poznan.pue.dit.zp.rm` package are presented in Fig. B.9. Classes of the RM module implement mechanism for recommendation monitoring. In particular they implement Recommendation Monitor component from Fig. 5.2 which is responsible for management of confidence indicator and social coefficient values. Confidence indicator and social coefficient values are used during recommendation formulation and ordering of recommendation elements (*cf.* Section 5.5.3).

Recommended activity pattern is transformed to recommendation rules by a parser modeled by `AbstractConforormanceRuleParser` class (*cf.* Section 5.6). Each rule is modeled by the `Rule` class. Access to rule logic is provided by `getLogic` method. This method is used by objects `of` the `RuleValidator` class. The `RuleValidator` class has a reference to `AbstractConformanceRuleParser` and uses `getRules` method of this class to retrieve generated rules. The `RuleValidator` class validates partial trace of running collaborative process instance using the `validateTrace` method. A result of validation is modeled by `RuleValidation` class. Each object of `RuleValidation` class has: one assigned rule, and one rule status modeled by `RuleStatus` class. Rule validation results are used to update confidence indicator value. Recommendation monitor represented by `RecommendationMonitor` class manages values of confidence indicators and social coefficients. The `getConfidanceIndicator` and `getSocialCoefficient` methods return up-to-date values of these metrics. The `RecommendationMonitorDao` class models the persistence layer of the module.

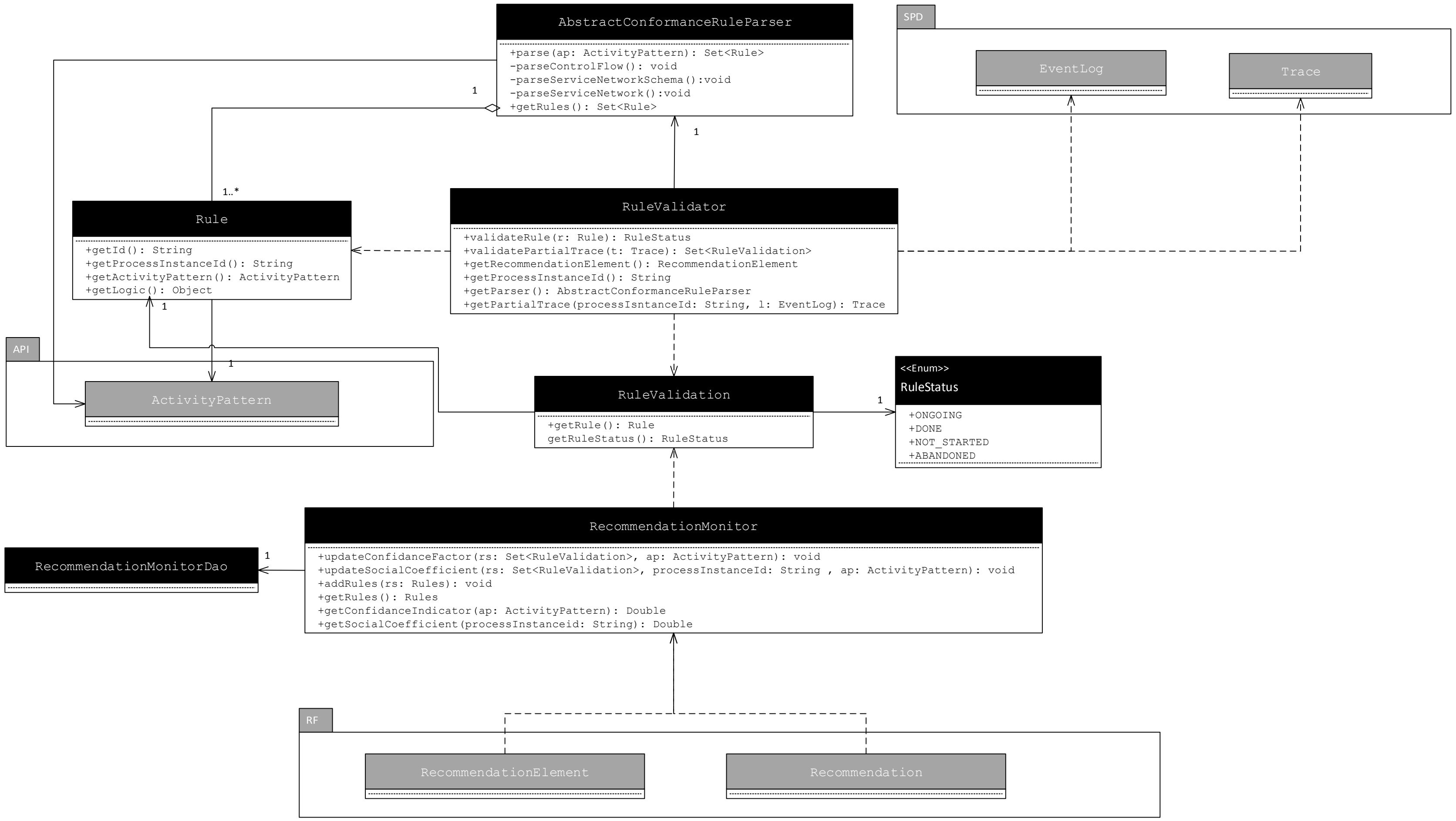

AbstractConformanceRuleParser
+parse(ap: ActivityPattern): Set<Rule>
-parseControlFlow(): void
-parseServiceNetworkSchema():void
-parseServiceNetwork():void
+getRules(): Set<Rule>
SPD
EventLog
Trace
1
1
1..*
Rule
+getId(): String
+getProcessInstanceId(): String
+getActivityPattern(): ActivityPattern
+getLogic(): Object
RuleValidator
+validateRule(r: Rule): RuleStatus
+validatePartialTrace(t: Trace): Set<RuleValidation>
+getRecommendationElement(): RecommendationElement
+getProcessInstanceId(): String
+getParser(): AbstractConformanceRuleParser
+getPartialTrace(processIsntanceId: String, l: EventLog): Trace
1
API
1
ActivityPattern
RuleValidation
+getRule(): Rule
getRuleStatus(): RuleStatus
1
<<Enum>>
RuleStatus
+ONGOING
+DONE
+NOT_STARTED
+ABANDONED
RecommendationMonitor
+updateConfidanceFactor(rs: Set<RuleValidation>, ap: ActivityPattern): void
+updateSocialCoefficient(rs: Set<RuleValidation>, processInstanceId: String , ap: ActivityPattern): void
+addRules(rs: Rules): void
+getRules(): Rules
+getConfidanceIndicator(ap: ActivityPattern): Double
+getSocialCoefficient(processInstanceid: String): Double
RecommendationMonitorDao
1
RF
RecommendationElement
Recommendation

Fig. B.9. Classes in Activity Pattern Instantiation module

Context and Function packages

Classes from `pl.poznan.pue.dit.zp.context` package are presented in Fig. B.10. All the classes:

- Model the concept of event context,
- Model the concept of activity instance context,
- Model the concept of activity pattern context and context classifier.

The `EventContext` class models the concept of event context. The access to the event context attributes is provided by `getAttributes` and `addAttribute` methods. The `ActivityInstanceContext` class holds reference to `EventContext` object that is returned by `getEventContext` method. Additionally, its methods permit accessing all the elements of activity instance context that follows formal model presented in Section 5.4.2, i.e., a set of events proceeding particular activity instance is returned by `getHistoryTrace` method, the `getProcessInstanceAttributes` method returns attributes of a process instance, the `getParallelEvent` method return events recorded at the time of activity instance. Finally, the `getHistoryServiceEntities` method returns service entities associated with proceeding events. `ActivityPatternContext` and `ContextClass` classes are extensions of `ActivityInstanceContext` class. The `validate` method of the `ContextClass` calculates a context distance between context class and the given context. A function being an instance of `AbstractClass` and provided by a user as a part of the RMV prototype configuration is used.

Each function provided by a user as a part of the RMV prototype configuration is an extension of `AbstractFunction` class from `pl.poznan.pue.dit.zp.functions` package (Fig. B.10). All the user-defined functions overload the implementation of the `evaluate` method.

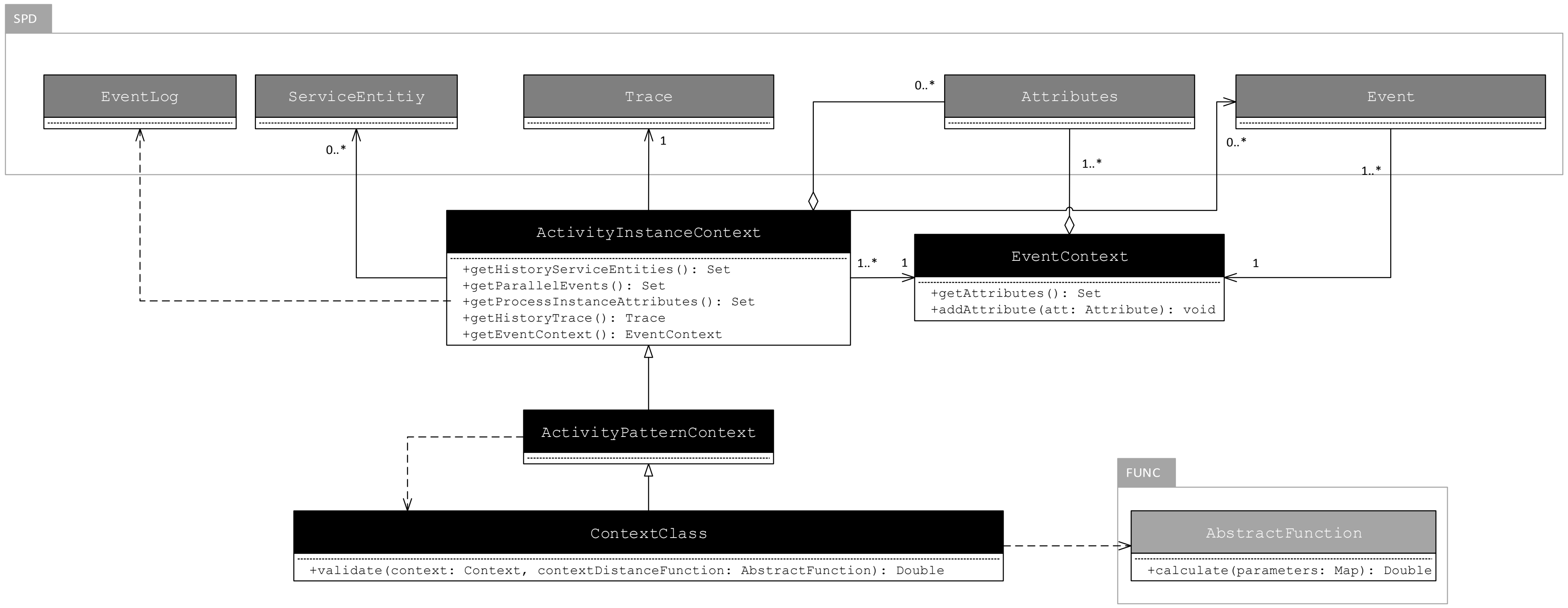

SPD
EventLog
ServiceEntitiy
Trace
Attributes
Event
0..*
1
0..*
1..*
0..*
1..*
ActivityInstanceContext
+getHistoryServiceEntities(): Set
+getParallelEvents(): Set
+getProcessInstanceAttributes(): Set
+getHistoryTrace(): Trace
+getEventContext(): EventContext
1..*
1
EventContext
+getAttributes(): Set
+addAttribute(att: Attribute): void
1
ActivityPatternContext
ContextClass
+validate(context: Context, contextDistanceFunction: AbstractFunction): Double
FUNC
AbstractFunction
+calculate(parameters: Map): Double

Fig. B.10. Context classes in Recommendation Formulation module

Appendix C. Table of Symbols Used in Dissertation

Objects and classes

a	attribute
an	attribute name
av	attribute value
$a_i \bullet a_j$	attribute equality
ob	object
$\underline{ob}$	object classifier
$ob_i \overset{ob}{=} ob_j$	equality according to classifier $\underline{ob}$
a^α	attribute constraint
ac	name of attribute constraint
ϑ_{ac}	attribute constraint predicate
$a \succ a^\alpha$	satisfaction of attribute constraint
c	class of an object
cn	name of class of an object
as^α	a set of attribute constraints
$ob \propto c$	object being instance of a class

Activities and processes

v	activity
V^*	denotes the set of all the activities
vd	activity description
VD^*	the set of all the activity descriptions
vi	activity instance
VI^*	the set of all the activity instances
vid	activity instance description
VID^*	the set of all the activity instance descriptions
p	process instance
pid	process instance description
π^α	process model
ar	actor
AR	the set of all the actor
ard	actor description
s	service
sq	service interface

Service protocols

se	service entity description
se^α	class of service entity description
SE^α	set of classes service entity description
σ	service network

σ_{SOVOBE}	the service network of all the members of SOVOBE
s_d	service description
S_d	the set of all the service descriptions
se^{α}	class of service consumer description
SC^{α}	the set of all classes of service consumer descriptions
si^{α}	class of service interface description
SI^{α}	the set of all classes of service interface descriptions
sp^{α}	class of service provider description
SP^{α}	the set of all classes of service provider descriptions
$S*^{\alpha}$	$SC^{\alpha} \cup SI^{\alpha} \cup SP^{\alpha}$
π_{sos}	service-oriented summary of a process model
l^{α}	service requirement
S_d	set of service descriptions
ρ	function mapping activity descriptions in π^{α} to service descriptions
σ^{α}	service network schema
Y^{α}	service protocol
$\overset{\alpha}{\Lambda}$	function mapping relations between the service-oriented summary of a process model and service network schema
Y^{β}	prototype service protocol
Ω	function associating service entity descriptions of the service network to elements of service descriptions
Φ	function associating nodes of service network to classes of service entity descriptions of the service network schema
Y^{ε}	executable service protocol
$\sigma \oplus \sigma^{\alpha}$	compliance with service network schema

Organizations

o	organization
η	organization goal
M	a set of organization members
R	relations among members of organization
O	the set of all the organizations
vo	virtual organization
SI	a set of service interface descriptions used by members of virtual organization
VO	the set of all the virtual organizations

Events

e	event
$\sum$	universe of events
E^*	the set of all the events
$E(vi_i)$	set of events assigned to activity insistence vi_i.
τ	process instance trace
$\tau(p,e)$	a subsequence of events from trace $\tau(p)$ that were recorded before event *e* inclusive
ℓ	event log
$A(\tau_p)$	a set of process instance attributes
$E(e,t_1,t_2)$	a set of events different than *e* that were recorded in an event log in time period from t_1 to t_2 inclusive
$e(id)$	the unique identifier of the event;
$e(processId)$	the unique identifier of the VO collaborative process;
$e(activityId)$	a unique identifier of the activity description ;
$e(processInstanceId)$	a unique identifier of the VO collaborative process instance;
$e(activityInstanceId)$	a unique identifier of the activity instance;
$e(time)$	*timestamp* of event *e*;
$e(trans)$	*type of* event *e* (for instance: start, resume, suspend, complete);
$e(context)$	the event context attribute;
$e(SC)$	a unique identifier of a service consumer;
$e(SP)$	a unique identifier of a service provider;
$e(SI)$	a unique identifier of a service interface;
$e(serviceEntity-SC)$	a service consumer description;
$e(serviceEntity-SP)$	a service provider description;
$e(serviceEntity-SI)$	a service interface description;
$e(social-SC)$	an event social attribute referring to service consumer;
$e(social-SP)$	an event social attribute referring to service provider;
$e(social-SI)$	an event social attribute referring to service interface.

Context

cf	context feature
CF	the set of all the context features
ce	context element
$ce(t)$	an event context element captured at time *t*
$co(t)$	context
CO^*	the set of all the contexts

$co(t, cf)$	a subset of context elements of context $co(t)$ where context element name is cf
$co(e)$	event context
sce	event social context element
$SE(e)$	a set of descriptions of service consumers and service providers who were involved in process instance execution before the event e inclusive
$co(vi)$	activity instance context
ce^{α}	context element constraint
sce^{α}	social context element constraint
co^{α}	context class
sco^{α}	social event context class
$class(co)$	function mapping context co into context class co^{α}
$eval(co, co^{\alpha})$	context distance

Activity patterns

$e_{s,vi}$	event corresponding to starting of activity instance vi execution
$e_{e,vi}$	event corresponding to completion of activity instance vi
G_p	temporal pattern of process instance p
Y_p	activity graph of process instance p
$Y' \pm \mathrm{Y}$	activity graph Y' is supported by activity graph Y
$PatternSearch\ (\tau(p))$	a function that discovers a set of activity graphs from collaborative trace $\tau(p)$
z	activity pattern
Z	a set of activity patterns
$sup(z)$	a function mapping activity patterns z to its support in event log ℓ
CMZ	a set of closed activity patterns
$co(z)$	activity pattern context
$co(\tau)$	a context of the activity instance completed as the last one in partial trace τ
qss	activity instance description sequence

Recommendation and instantiation

recom_index	recommendation index
nonrecom_index	non-recommendation index
$\delta(\tau, z)$	recommendation element
$\aleph$	recommendation justification
RQ	recommendation query
$R(\tau)$	recommendation
$compliance(\sigma, \sigma^{\alpha})$	a function mapping a service network σ and service network schema σ^{α} to compliance value

POLITECHNIKA GDAŃSKA
Wydział Elektroniki, Telekomunikacji
i Informatyki

Zbigniew Paszkiewicz

Metoda rekomendacji RMV dla doboru partnerów i usług w inkubatorach wirtualnych organizacji oparta na technikach eksploracji procesów

Streszczenie rozprawy

Promotor:
prof. dr hab. inż. Wojciech Cellary
Wydział Informatyki i Gospodarki Elektronicznej
Uniwersytet Ekonomiczny w Poznaniu

Gdańsk, 2014

1. Wstęp

W gospodarce otoczenie organizacji w dużym stopniu wpływa na sposób ich funkcjonowania (Stoner, et al., 1999). Współczesne trendy: globalizacja, rozwój i upowszechnienie technologii informacyjnych, rozwój elektronicznej gospodarki opartej na wiedzy i rosnąca konkurencja rynkowa, przekładają się na złożoność, niepewność, dynamizm, burzliwość i zróżnicowanie otoczenia organizacji. W takim otoczeniu produkcja i świadczenie usług na skalę światową wymaga dużych zasobów i zróżnicowanych kompetencji, których pojedyncza organizacja zwykle nie posiada. Nowoczesne organizacje wcielają zatem w życie strategie sprzyjające integracji i współpracy wielu różnorodnych, wyspecjalizowanych, autonomicznych jednostek posiadających komplementarne zasoby i kompetencje (Porter, 2008).

W wyniku studiów nad współpracą międzyorganizacyjną zaproponowano pojęcie *wirtualnej organizacji* (VO, ang. *Virtual Organization*) *jako „sieci wielu różnorodnych aktorów, nazywanych członkami wirtualnej organizacji, którzy są w znacznym stopniu autonomiczni, rozproszeni geograficznie i heterogeniczni pod względem kultury organizacyjnej, kapitału społecznego, celów i otoczenia, w których działają. Członkowie wirtualnej organizacji wspólnie realizują co najmniej jeden proces współpracy wirtualnej organizacji, którego celem jest zaspokojenie potrzeby klienta wirtualnej organizacji"* (Camarinha-Matos, et al., 2008). Elastyczne powiązania pomiędzy członkami VO oparte na technologiach informatycznych, zwiększają zwinność współpracy między nimi i pozwalają na lepsze radzenie sobie ze złożonością otoczenia biznesowego. Członkami VO są organizacje – przedsiębiorstwa, jednostki administracji publicznej i/lub organizacje pozarządowe – oraz ludzie i systemy informatyczne.

Sukces VO w dużym stopniu zależy od efektywności i skuteczności współpracy między ich członkami. Kluczowe znaczenie ma tu właściwy dobór członków VO. Ze względu na złożoność tego problemu, w literaturze naukowej przedmiotu zaproponowano wiele metod wsparcia doboru członków VO zarówno wsparcia informatycznego, jak i organizacyjnego. W tym kontekście zaproponowano pojęcie Inkubatora Wirtualnych Organizacji (VOBE, ang. *Virtual Organization Breeding Environment*) jako *„stowarzyszenia organizacji, którego celem jest podniesienie ogólnego poziomu przygotowania członków tego stowarzyszenia do potencjalnej, przyszłej współpracy w ramach wirtualnej organizacji"* (Camarinha-Matos, et al., 2008). VOBE pozwala na wstępne przygotowanie się danej organizacji do współpracy z innymi członkami VOBE zanim pojawi się konkretna okazja biznesowa wymagająca utworzenia wirtualnej organizacji. Przygotowanie odbywa się przez ustalenie wspólnych standardów współdzielenia informacji, procedur postępowania, sposobów współdzielenia infrastruktury informatycznej itp. VOBE, którego działanie i infrastruktura techniczna są zorganizowane zgodnie z architekturą usługową SOA (OASIS Technical Committee, 2006), nazywa się *Usługowym Inkubatorem Wirtualnych Organizacji* (SOVOBE, ang. *Service-Oriented Virtual Organization Breeding Environment*) (Picard, et al., 2010). W SOVOBE, interakcje między członkami VO, wirtualnymi organizacjami i infrastrukturą SOVOBE odbywają się w formie świadczenia usług.

Złożona struktura VO przekłada się na złożoność procesu współpracy między członkami VO. Dwiema głównymi cechami procesu współpracy w VO są: nieprzewidywalność (ang.

unpredictability) i wyłanianie się (ang. *emergence*). *Nieprzewidywalność* procesu współpracy w VO odnosi się do trudności przewidzenia z wyprzedzeniem przebiegu wykonania instancji procesu współpracy w VO. *Wyłanianie się* procesu współpracy w VO odnosi się do wpływu, jaki ma częściowe wykonanie instancji procesu współpracy w VO na jej dalszy przebieg. Przykładowo, decyzje podjęte we wstępnym etapie wykonania instancji procesu współpracy w VO mają wpływ na zbiór czynności wykonywanych w jej dalszych etapach. Także cel wykonania instancji procesu współpracy w VO może ulec przeformułowaniu w trakcie jej wykonania. Z cech nieprzewidywalności i wyłaniania się instancji procesu współpracy w VO wynika *wysoki poziom nieustrukturyzowania* procesu współpracy w VO.

W konsekwencji upowszechnienia się technologii informacyjnych i wszechobecnego dostępu do internetu za pomocą urządzeń stacjonarnych i mobilnych, duża liczba procesów jest wykonywanych przy wykorzystaniu technologii informatycznych. Pojęcie *systemów zorientowanych procesowo* (ang. *Process-Aware Information Systems*) zostało zaproponowane jako ogólne określenie systemów informatycznych wspierających różne fazy zarządzania procesami (Aalst, 2004) (Reichert, 2011). Systemy zorientowane procesowo są *„systemami informatycznymi, które wspierają zarządzanie procesami na wszystkich etapach jego cyklu życia procesu: budowy modelu, konfiguracji i implementacji instancji procesu, uruchomienia instancji procesu, monitorowania i modyfikacji instancji procesu, gdzie wykonanie instancji procesu wymaga zaangażowania ludzi, systemów informacyjnych i wykorzystania różnych źródeł informacji”* (Dumas, et al., 2005). Przykładami systemów zorientowanych procesowo są systemy przepływu prac (ang. *workflow*) i systemy wsparcia współpracy (ang. *Computer Supported Collaborative Work (CSCW)* systems). Projektanci systemów zorientowanych procesowo dążą do równowagi pomiędzy oferowanym przez system zorientowany procesowo wsparciem użytkownika w wyborze czynności do realizacji i w wykonaniu tych czynności, a elastycznością w modyfikowaniu przebiegu instancji procesu. Systemy pozwalające na większą elastyczność oferują mniejsze wsparcie dla wykonywanych czynności i *vice versa*. Typowe jest, że oferowane przez systemy zorientowane procesowo wsparcie dla wykonania czynności wykorzystuje predefiniowane, z góry ustalone modele procesów, które szybko ulegają dezaktualizacji w dynamicznym środowisku organizacji.

Nowoczesne systemy zorientowane procesowo zapisują duże ilości szczegółowych danych o przebiegu wykonania instancji procesów i czynności w formie tzw. *dzienników zdarzeń*. Analiza dzienników zdarzeń pozwala na odkrycie czynników wpływających na efektywność wykonania instancji procesów. Wiedza na temat tych czynników może być wykorzystania do poprawy efektywności wykonania przyszłych instancji procesów. Skuteczne wykorzystanie danych z dzienników zdarzeń zależy od efektywności metod ich analizy i zdolności tych metod do formułowania wniosków dotyczących wykonań procesów na poziomie biznesowym.

Odkrywanie wiedzy na podstawie dużych ilości danych jest przedmiotem takich obszarów badawczych w informatyce jak eksploracja danych i uczenie maszynowe. Nowe pojęcie *eksploracji procesów* zostało wprowadzone jako określenie na podzbiór metod eksploracji danych, które bazują na pojęciu procesu. *Eksploracja procesów* jest *„zbiorem technik, narzędzi i metod pozwalających na odkrywanie, monitorowanie i ulepszanie rzeczywistych procesów biznesowych na podstawie wiedzy wyekstrahowanej z dzienników zdarzeń dostępnych w nowoczesnych systemach zorientowanych procesowo”* (Aalst, 2011). W badaniach nad eksploracją procesów wyróżnia się dwie główne grupy metod: metody analizy *off-line* i metody analizy *on-line*. Grupa metod analizy *off-line* obejmuje metody *budowy modeli procesów*, metody *weryfikacji modeli procesów* i metody *rozbudowy modeli procesów*. Grupa metod analizy *on-line* obejmuje metody *rozpoznania, predykcji* i *rekomendacji* (Aalst, 2011).

Efektywne wykorzystanie metod rekomendacji *on-line* opartych na eksploracji procesów pozwala na podniesienie efektywności wykonania procesów wspomaganego przez systemy zorientowane procesowo (Aalst, et al., 2010). Metody te polegają na analizie zakończonych i trwających wykonań instancji procesów w celu odkrycia ich rzeczywistych (a nie założonych, teoretycznych) modeli, a następnie wykorzystaniu tych modeli do wsparcia użytkownika przez system zorientowany procesowo w formie rekomendacji kolejnych czynności do wykonania.

Rekomendacja czynności oparta na analizie dzienników zdarzeń jest nowym obszarem badań naukowych – pierwsze prace na ten temat pojawiły się w roku 2008 (Schonenberg, et al., 2008). Istniejące metody rekomendacji czynności oparte na eksploracji procesów ograniczającą się do procesów ustrukturyzowanych o znanych a priori modelach. Takie metody nie mogą być zastosowane do wsparcia ważnego zbioru procesów, które są nieustrukturyzowane, nieprzewidywalne i wyłaniające się. Ponadto, istniejące metody rekomendacji bazują tylko na odkrywaniu nazw czynności, analizie ich uporządkowania i ich charakterystykach czasowych. Metody te nie uwzględniają wielu aspektów ważnych z punktu widzenia efektywnego doboru partnerów i usług do VO, w tym informacji o aktorach biorących udział w wykonaniu poszczególnych czynności i kontekście wykonania poszczególnych instancji procesów. Potrzeba opracowania metod rekomendacji czynności wspierających wykonanie nieustrukturyzowanych procesów współpracy w VO została podkreślona w (Sinur & Jones, 2012): *„w procesach pół-ustrukturyzowanych i nieustrukturyzowanych, w których istnieje wiele sposobów wykonania każdej instancji procesu, czynniki warunkujące sukces ich wykonania mogą być zidentyfikowane przy wykorzystaniu metod automatycznego budowania modeli procesów. Następnie, odkryte czynniki sukcesu mogą być prezentowane aktorom uczestniczącym w przyszłych realizacjach procesów jako przykłady dobrych praktyk, uszeregowane według celów osiąganych przy ich wykorzystaniu. Głównymi obszarami zastosowań dla takich metod są: zwinne zarządzanie przypadkami (ang. adaptive case management), procesy współpracy i interakcje społeczne”.*

Metoda rekomendacji dla wirtualnych organizacji RMV (ang. *Recommendation Method for Virtual Organizations*) przedstawiona w tej rozprawie polega na automatycznym odkrywaniu szablonów czynności i generowaniu rekomendacji ad-hoc dla procesów współpracy w VO wykonywanych w ramach SOVOBE. Szablon czynności jest częściowo uporządkowanym zbiorem czynności, który często występuje w wielu instancjach procesów współpracy w VO. W metodzie RMV zakłada się, że dziennik zdarzeń systemu zorientowanego procesowo gromadzi zdarzenia spełniające następujące trzy wymagania:

1. każde zdarzenie w dzienniku zdarzeń ma zbiór atrybutów opisujących członków VO zaangażowanych wykonanie instancji czynności powiązanej z tym zdarzeniem;
2. każde zdarzenie w dzienniku zdarzeń ma zbiór atrybutów opisujących okoliczności, w jakich zostało zarejestrowane;
3. każda zakończona instancja procesu współpracy w VO jest opisana przez zbiór atrybutów, z których jeden opisuje wynik wykonania tej instancji procesu współpracy w VO.

Metoda RMV składa się z dwóch faz:

1. odkrywania szablonów czynności i ich kontekstów,
2. formułowania rekomendacji.

W pierwszej fazie, zbiór szablonów czynności jest odkrywany na podstawie analizy danych z dziennika zdarzeń. Każdy szablon czynności zawiera informację o kontekstach, w jakich wystąpił, częściowo uporządkowany zbiór czynności, zbiór członków VO zaangażowanych w wykonanie czynności składających się na szablon czynności i zbiór relacji społecznych występujących między tymi członkami VO. W drugiej fazie metody RMV wykonywanej na żądanie użytkownika, szablon czynności pasujący do kontekstu wykonywanej i niezakończonej instancji procesu współpracy w VO jest rekomendowany do uwzględniania podczas kontynuacji jej wykonywania współpracy w VO. Przed włączeniem w wykonanie instancji procesu współpracy w VO, szablon czynności jest instancjonowany. Podczas instancjonowania są wykorzystywane: (1) informacje zawarte w szablonie czynności dotyczące pożądanych cech członków VO i relacji między nimi, (2) informacje o członkach SOVOBE, do których dostęp jest możliwy za pomocą usług świadczonych przez SOVOBE. Wybór szablonu czynności do wykorzystania w instancji procesu współpracy w VO i instancjonowanie szablonu czynności są również wykonywane na drodze współpracy. Rekomendacje generowane przez metodę RMV są wykorzystywane przez system zorientowany procesowo do wsparcia uczestników instancji procesu współpracy w VO.

Teza rozprawy jest sformułowana następująco:

Metoda RMV pozwala na trafne formułowanie rekomendacji prowadzących do wyznaczenia zbioru partnerów i usług do współpracy w ramach wirtualnej organizacji.

2. Komputerowe wsparcie procesów współpracy w VO

Badania nad współpracą ludzi, organizacji i systemów informatycznych są równolegle prowadzone przez dwie społeczności naukowe: (1) społeczność organizacyjnych sieci współpracy (Porter, 2008) (Camarinha-Matos, et al., 2007) (Rabelo & Gusmeroli, 2008) (Cellary & Strykowski, 2009) (Picard & Cellary, 2010) i (2) rozproszonych systemów informatycznych (OASIS Technical Committee, 2006) (Workflow Management Coalition, 1999). Społeczności te zaproponowały różne podejścia do opisu organizacji (Gallon, et al., 1995) (Ermilova & Afsarmanesh, 2010), opisu systemów informatycznych, opisu usług świadczonych przez organizacje i systemy informatyczne (Claro, et al., 2005) (Jaeger & Mühl, 2007). Zaproponowano także struktury organizacyjne wspierające współpracę ludzi, organizacji i systemów informatycznych (Camarinha-Matos, et al., 2007). W literaturze zaproponowano szereg podejść do rozwiązania problemu efektywnego doboru partnerów i usług do procesów współpracy (Jarimo, 2009) (Ding, et al., 2003) (Canfora, et al., 2005) (Crispim & Sousa, 2007) (Rabelo & Gusmeroli, 2008).

Wśród metod komputerowego wsparcia procesów współpracy w VO na wyróżnienie zasługują wspominane we wstępie (*cf.* Rozdział 0) systemy zorientowane procesowo oraz systemy rekomendacji kontekstowych (ang. *context-aware recommender systems*). Te drugie (Adomavicius, et al., 2005) (Adomavicius & Tuzhilin, 2005) (Ricci, et al., 2011) opierają się na pojęciu kontekstu (Dey, 2001) (Tan, et al., 2008) (Abowd, et al., 1999), który jest podstawą do generowania rekomendacji czynności dla użytkowników systemów rekomendacyjnych.

Szczególnym typem systemów rekomendacyjnych są systemy wykorzystujące techniki eksploracji procesów. Metody eksploracji procesów pozwalają na odkrycie i analizę rzeczywistych zachowań użytkowników w konkretnych sytuacjach praktycznych, zamiast bazować na domniemanych, założonych teoretycznie zachowaniach ludzi (Aalst, et al., 2010) (Schonenberg, et al., 2008) (Nakatumba, et al., 2012) (Haisjackl & Weber, 2011) (Dorn, et al., 2010) (Swinkels, 2012).

Eksploracja procesów nie jest jedyną techniką odkrywania zachowań ludzkich na podstawie analizy dużych ilości danych. Na gruncie eksploracji danych opracowano metody odkrywania wzorców sekwencji (Agrawal & Srikant, 1995) (Srikant & Agrawal, 1996) (Mane, 2013) (Wang & Han, 2004) (Hwang, et al., 2004), klasyfikacji (Witten, et al., 2011) i grupowania (Delias, et al., 2013). Metody te jednak nie radzą sobie jednak z odkryciem podstawowych konstrukcji wykorzystywanych w modelach procesów, jak np. równoległe wykonanie czynności, pętle i miejsca decyzyjne. Metody analizy sieci społecznych (Morzy & Forenc, 2013) (Brendel & Krawczyk, 2010) (Watts, 2004) abstrahują od pojęć czynności i procesu.

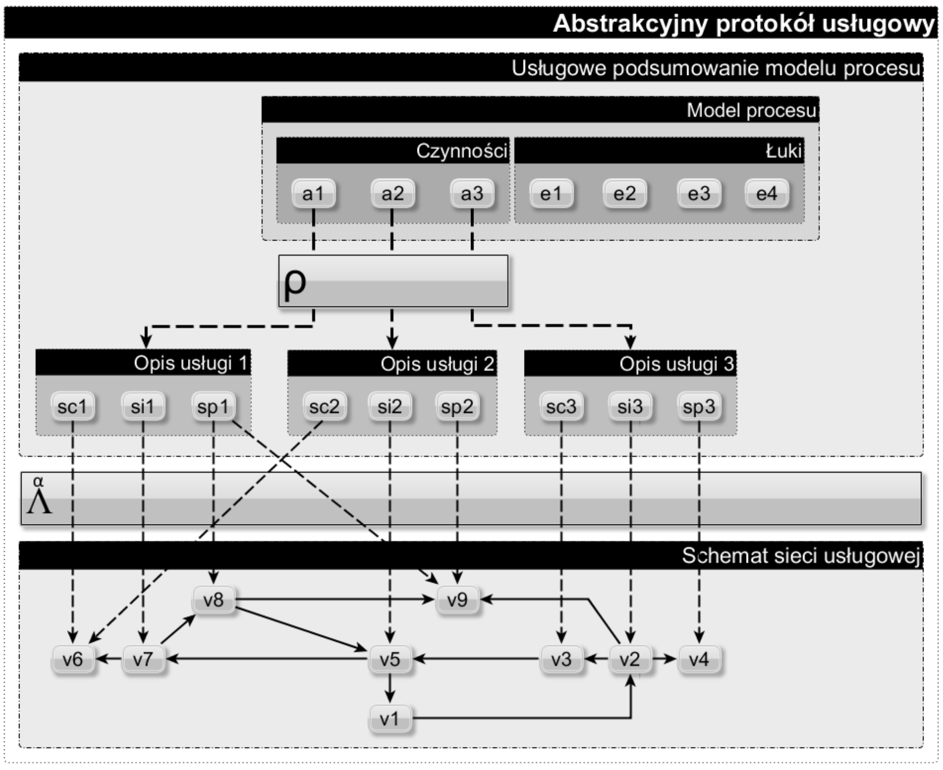
Istniejące metody komputerowego wsparcia modelowania procesów, wykonania instancji procesów i odkrywania zachowań ludzkich są niewystarczające dla efektywnego wsparcia procesów współpracy w VO. Systemy zorientowane procesowo ograniczają elastyczność planowania współpracy lub oferują ograniczone wsparcie dla planowania i wykonania czynności. Metody rekomendacji czynności opierają się statycznych, predefiniowanych modelach działań użytkowników i modelach kontekstu, które szybko ulegają dezaktualizacji w dynamicznym otoczeniu organizacji. Dynamika zmian kontekstu jest modelowana w ograniczonym stopniu. Rekomendacje generowane przez metody oparte na metodach eksploracji danych, w tym eksploracji procesów, dotyczą zwykle tylko czynności, a pomijają takie aspekty jak cechy współpracujących partnerów i cechy relacji między nimi. Aspekty te są ważne dla efektywnego doboru partnerów do procesów współpracy w VO. Wreszcie w istniejących metodach doboru partnerów i usług zakłada się pełną wiedzę o modelu procesu współpracy w VO. Dobór partnerów i usług wykonywany przy ich użyciu jest wykonywany tylko raz przed rozpoczęciem realizacji instancji procesu współpracy w VO. Tymczasem w praktyce, w dynamicznym otoczeniu, takie podejście nigdy nie jest stosowane. Ponadto istniejące metody doboru partnerów i usług nie biorą pod uwagę aspektów społecznych, które w praktyce mają znaczący wpływ na ogólną zdolność do współpracy członków VO.

Istnieje zatem potrzeba opracowania nowej metody wspierania instancjonowania i wykonania procesów współpracy w VO, która będzie maksymalizować wsparcie dawane użytkownikowi przy jednoczesnym małym negatywnym wpływie na elastyczności przebiegu instancji procesu współpracy w VO. Metoda ta nie będzie wymagać apriorycznej wiedzy o modelu procesu współpracy w VO oraz będzie uwzględniać ważne dla procesów współpracy w VO aspekty kontekstowe i społeczne, które pozwolą na efektywny dobór partnerów i usług dokonywany przez cały okres realizacji procesu współpracy w VO.

3. Protokoły usługowe

W (Picard, 2013) przeanalizowano różne podejścia do modelowania interakcji międzyludzkich. Na podstawie wyników tej analizy zaproponowano pojęcie *protokołu usługowego* jako podejścia do modelowania i adaptacji interakcji pomiędzy ludźmi, organizacjami i systemami informatycznymi w zmiennym otoczeniu. W metodzie RMV protokoły usługowe wykorzystywane są jako element reprezentacji szablonów czynności.

Protokół usługowy składa się z czterech elementów: (1) modelu procesu, (2) usługowego podsumowania modelu procesu, (3) sieci usługowej, oraz (4) schematu sieci usługowej. *Model procesu* jest zbiorem częściowo uporządkowanych czynności, gdzie relacją porządkującą jest czasowe następstwo wykonywanych czynności. *Usługowe podsumowanie modelu procesu* (ang. *service-oriented summary of a process model*) zawiera powiązanie każdej czynności z modelu procesu z opisem usługi. *Opis usługi* składa się z trzech *elementów opisu usługi*: (1) *opisu konsumenta usługi* odpowiedzialnego za wykonanie czynności, (2) *opisu interfejsu usługi* wykorzystywanej przez konsumenta usługi do wykonania czynności, (3) *opisu dostawcy usługi* odpowiedzialnego za wykonanie usługi. Usługowe podsumowanie modelu procesu stanowi dodatkową warstwę abstrakcji pozwalającą na reprezentację czynności z modelu procesu w sposób zgodny z architekturą usługową SOA. Informacja o *jednostkach usługowych* (ang. *service entity*), czyli konsumentach usług, dostawcach usług i interfejsach usług, jest przechowywana w sieci usługowej. *Sieć usługowa* (ang. *service network*) jest źródłem informacji o cechach jednostek usługowych i cechach *relacji usługowych* występujących pomiędzy nimi. Zbiór wymagań wobec cech jednostki usługowej nazywa się *klasą jednostki usługowej* (ang. *class of service entity*). Klasy jednostek usługowych są powiązane z elementami opisów usług. Zbiór wymagań dotyczących relacji usługowych pomiędzy jednostkami usługowymi nazywa się *wymaganiem usługowym* (ang. *service requirement*). *Schemat sieci usługowej* (ang. *service network schema*) jest grafem, w którym wierzchołkami są klasy jednostek usługowych, a łukami wymagania usługowe. Schemat sieci usługowej ogranicza zbiór jednostek usługowych, które mogą wziąć udział w wykonaniu protokołu usługowego. Instancjonowanie protokołu usługowego polega na znalezieniu zbioru jednostek usługowych i przypisaniu ich do klas jednostek usługowych tak, aby ograniczenia zdefiniowane w klasach jednostek usługowych i w wymaganiach usługowych były spełnione w najwyższym możliwym stopniu. Protokół usługowy bez przypisanych jednostek usługowych jest nazywany protokołem *abstrakcyjnym*. Przykład protokołu abstrakcyjnego jest przedstawiony na Rys. 3.1.



Rys. 3.1. Abstrakcyjny protokół usługowy

Protokół usługowy jest nazywany protokołem *prototypowym,* jeśli tylko wybrane klasy jednostek usługowych mają przypisane jednostki usługowe. Natomiast protokół usługowy jest nazywany protokołem *wykonywalnym*, jeśli każda klasa jednostki usługowej ma przypisaną jednostkę usługową.

Protokoły usługowe mają zestaw cech, którym pozwalają na efektywne modelowanie procesów współpracy w VO:

- *oddzielenie implementacji czynności od modelu procesu* – jedna czynność może mieć wiele różnych implementacji różniących się między sobą konsumentami usług, interfejsami usługi lub dostawcami usług biorącymi udział w wykonaniu czynności;
- *modelowanie odpowiedzialności* – protokoły usługowe pozwalają na uchwycenie faktu odpowiedzialności klienta usługowego za wywołanie czynności i odpowiedzialności dostawcy usługi za wykonanie usługi;
- *modelownie wymagań dotyczących członków VO i relacji usługowych* – protokoły usługowe umożliwiają definiowanie wymagań dotyczących współpracujących członków VO i relacji między nimi.

Model formalny protokołów współpracy jest przedstawiony w (Picard, 2013).

4. Metoda RMV

4.1. Wymagania dla metody RMV

Na podstawie analizy stanu wiedzy (*cf.* Rozdział 2 rozprawy) zdefiniowano następujące wymagania dla metody efektywnego wsparcia wykonania procesów współpracy w VO:

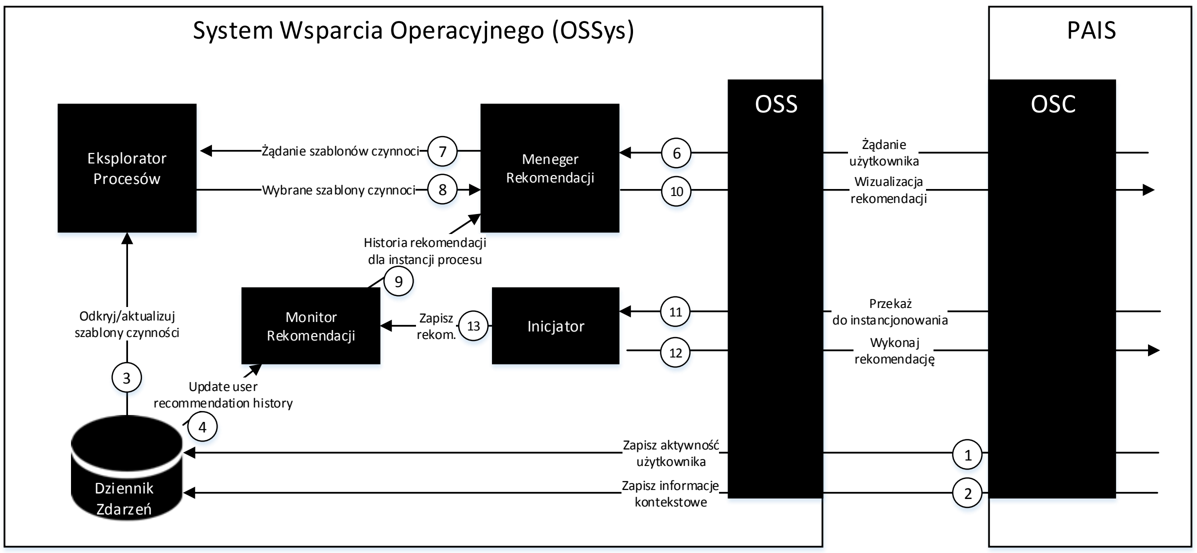
1. wsparcie uczestników procesu współpracy w VO powinno mieć formę wytycznych dotyczących koniecznych do wykonania czynności i wytycznych dotyczących członków VO, którzy powinni być zaangażowani w wykonanie tych czynności; jeśli członkowie VO nie mogą być imiennie wskazani, to należy wskazać ich pożądane cechy;
2. wsparcie uczestników procesu współpracy w VO powinno obejmować mechanizm weryfikacji zgodności realizacji instancji procesu współpracy w VO z przyjętymi wytycznymi; odstępstwa od przyjętych wytycznych powinny być podstawą do zmiany wytycznych lub uzyskania nowych;
3. kluczowe jest wsparcie dla adaptacji i elastyczności wykonania instancji procesu współpracy w VO przez reagowanie na zmiany w otoczeniu, w jakim instancja procesu współpracy w VO jest wykonywana współpracy w VO; zmiana w otoczeniu powinna skutkować uzyskaniem nowych wytycznych dotyczących wykonania instancji procesu VO dopasowanych do tego nowego otoczenia;
4. wsparcie powinno opierać się na analizie faktycznych, rzeczywistych zachowań uczestników procesu współpracy w VO, a nie na predefiniowanych modelach procesów, które szybko ulegają dezaktualizacji w dynamicznym otoczeniu;
5. celem jest komputerowe wsparcie współpracujących członków VO w podejmowaniu decyzji dotyczącej wykonania procesu współpracy w VO, a nie zastąpienie ich mechanizmem automatycznego podejmowania decyzji;
6. wytyczne dotyczące wykonania procesu współpracy w VO powinny bazować na tzw. *mądrości tłumu* (ang. *collaborative wisdom*), czyli powinny opierać się na obserwacji wielu różnych wykonań procesów współpracy w VO i uwzględniać różne możliwości wykonania instancji procesów;
7. wyniki działania metody wsparcia procesów współpracy w VO powinny nadawać się do wielokrotnego wykorzystania w różnych instancjach procesów współpracy w VO, w różnych VO;
8. wsparcie procesów współpracy w VO musi uwzględniać aspekty społeczne i kontekstowe mające kluczowy wpływ na współpracę w ramach VO;
9. dobór członków VO do procesu współpracy w VO musi być realizowany konsekwentnie przez cały czas wykonania instancji procesu współpracy w VO.

4.2. Koncepcja metody RMV

Metoda rekomendacji dla wirtualnych organizacji RMV (metoda RMV, ang. *Recommendation Method for Virtual Organizations*) polega na automatycznym odkrywaniu szablonów czynności i generowaniu rekomendacji ad-hoc dla procesów współpracy w VO wykonywanych w ramach SOVOBE. *Szablon czynności* jest protokołem usługowym, który jest często wykonywany w określonych kontekstach jako część instancji procesów współpracy w VO. Odkryte szablony czynności są wykorzystywane jako rekomendacje czynności i członków VO. Wybór szablonów czynności do rekomendacji opiera się na podobieństwie pomiędzy kontekstami szablonu czynności a aktualnym kontekstem instancji procesu współpracy w VO. Szablon czynności, który najlepiej pasuje do kontekstu instancji procesu współpracy w VO jest instancjonowany i włączany w wykonanie instancji procesu współpracy w VO. Wybór i instancjonowanie rekomendowanego szablonu czynności odbywa się na drodze współpracy grupy aktorów nazywanych *partnerami dobierającymi*.

Części składowe metody RMV są przedstawione na Rys. 4.1. Dwie główne składowe metody RMV to system zorientowany procesowo (PAIS, ang. *Process-Aware Information System*) i system wsparcia operacyjnego (OSSys, ang. *Operational Support System*). PAIS umożliwia współpracującym członkom VO wykonywanie czynności w ramach różnych instancji procesów współpracy w VO. Potencjalnie PAIS wspiera wykonanie wielu instancji procesów współpracy w VO równocześnie. OSSys stanowi faktyczną implementację metody RMV.

Dwa komponenty biorą udział w komunikacji pomiędzy PAIS i OSSys: Usługa Wsparcia Operacyjnego (OOS, ang. *Operational Support Service*) i Klient Wsparcia Operacyjnego (OSC, ang. *Operational Support Client*). OSC umożliwia korzystanie z funkcjonalności OSSys. OSC jest zintegrowany z PAIS i jest odpowiedzialny za wymianę danych pomiędzy SZP i UWO. Żądanie rekomendacji użytkownika PAIS jest przesyłane przez OSC do OSS. OSS przekazuje żądanie do poszczególnych komponentów OSSys. Odpowiedź wygenerowana przez komponenty OSSys jest przekazywana do OSS i dalej do OSC.


System Wsparcia Operacyjnego (OSSys)
PAIS
OSS
OSC
Eksplorator Procesów
Meneger Rekomendacji
Żądanie szablonów czynności
7
Wybrane szablony czynności
8
6
10
Żądanie użytkownika
Wizualizacja rekomendacji
Historia rekomendacji dla instancji procesu
9
Monitor Rekomendacji
Zapisz rekom.
13
Inicjator
11
12
Przekaż do instancjonowania
Wykonaj rekomendację
Odkryj/aktualizuj szablony czynności
3
Update user recommendation history
4
Dziennik Zdarzeń
Zapisz aktywność użytkownika
1
Zapisz informacje kontekstowe
2

Rys. 4.1. Główne składowe metody RMV

Metoda RMV składa się z dwóch faz:

1. *odkrycie szablonów czynności i ich kontekstów* – odkrycie szablonów czynności wymaga oddzielenia tych instancji procesu współpracy w VO, które są powtarzalne i wspólne dla wielu instancji procesów współpracy w VO, od tych, które są unikalne i niepowtarzalne; powtarzalne części instancji procesów współpracy w VO są przechowywane jako szablony czynności wraz z kontekstami, w których zostały zidentyfikowane;
2. *formułowanie rekomendacji* – w tej fazie, ze zbioru wszystkich szablonów czynności są wybierane szablony czynności dopasowane do kontekstu rozpatrywanej instancji

procesu współpracy w VO; szablony te są rekomendowane; jeden szablon wybrany do wykonania jest instancjonowany; faza druga jest wykonywane na żądanie użytkownika systemu zorientowanego procesowo; żądanie może być wygenerowane ręcznie przez użytkownika lub może być wysyłane do OSSys automatycznie za każdym razem, gdy PAIS wykryje zmianę stanu realizowanej instancji procesu współpracy w VO.

Pierwsza faza obejmuje kroki 1-3 widoczne na Rys. 4.1. Informacje o czynnościach wykonywanych przez współpracujących członków VO są zapisywane w *Dzienniku Zdarzeń* (krok 1). Zdarzenia zgromadzone w *Dzienniku Zdarzeń* są powiązane zarówno z zakończonymi, jak i jeszcze wykonywanymi instancjami procesów współpracy w VO. Każde zapisane zdarzenie zawiera informację o kontekście, w jakim wystąpiło (krok 2). *Eksplorator Procesów* analizuje dane przechowywanych w *Dzienniku Zdarzeń* i na ich podstawie odkrywa szablony czynności oraz ich konteksty (krok 3). Szablony czynności wraz z kontekstami są przechowywane w *Eksploratorze Procesów*. Odkrywanie szablonów czynności odbywa się w określonych odstępach czasu. Każde kolejne szablony czynności są odkrywane przy wykorzystaniu coraz większej ilości danych gromadzonych w *Dzienniku Zdarzeń*. Interwał czasowy pomiędzy kolejnymi odkryciami szablonów czynności jest jednym z parametrów metody RMV.

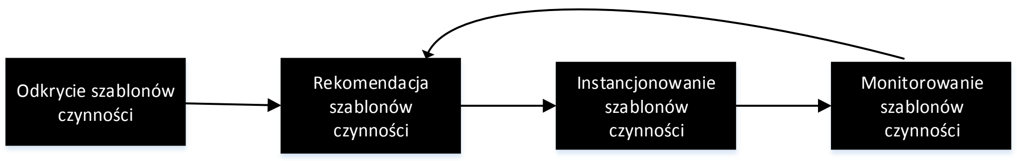
Druga faza metody RMV obejmuje kroki 4-13 widoczne na Rys. 4.1. Formułowanie rekomendacji obejmuje wybór szablonu czynności najlepiej pasującego do kontekstu instancji procesu współpracy w VO i instancjonowanie szablonu czynności. Te kroki wykonane dla wielu szablonów czynności w czasie całego wykonania procesu współpracy w VO tworzą *proces doboru*. Instancja procesu współpracy, dla którego są tworzone rekomendacje i jest prowadzony dobór nazywany jest procesem *dobieranym*. Procesy *doboru* i *dobierane* są ściśle powiązane – przebieg jednego procesu ma znaczący wpływ na przebieg drugiego. Członkowie VO współpracujący w ramach procesu doboru, czyli *partnerzy dobierający*, są z reguły inni niż członkowie VO współpracujący w ramach procesu dobieranego. Wygenerowanie rekomendacji ma miejsce w odpowiedzi na żądanie partnera dobierającego (krok 6). Żądanie zawiera aktualny kontekst procesu współpracy w VO i preferencje partnera dobierającego. Preferencje są wykorzystywane do sparametryzowania metody RMV. W pierwszej kolejności żądanie jest przekazywane do *Eksploratora Procesów* (krok 7), który dokonuje wyboru podzbioru szablonów czynności, które mogą być rekomendowane w konkretnym kontekście instancji procesu współpracy w VO (krok 8). Ostateczne uszeregowanie szablonów czynności jest wykonywane przez *Menedżer Rekomendacji*. Uszeregowanie szablonów czynności uwzględnia:

- zbiór szablonów czynności pochodzący od *Eksploratora Procesów*,
- informacje pochodzące od *Monitora Rekomendacji* (krok 9); *Monitor Rekomendacji* udostępnia informacje dotyczące historii rekomendacji wygenerowanych wcześniej dla tej instancji procesu współpracy w VO i informacje opisujące stopień wykorzystania rekomendacji przez współpracujących członków VO (krok 4).

Menedżer Rekomendacji prezentuje zbiór uszeregowanych szablonów czynności partnerom dobierającym (krok 10). Partnerzy dobierający dokonują ostatecznego wyboru jednego szablonu czynności, który zostanie włączony w wykonanie instancji procesu współpracy w VO. Ten szablon czynności jest wówczas przekazany do *Inicjatora* (krok 11), gdzie jest instancjonowany, a następnie przekazany do wykonania (krok 12). Informacja o decyzji dobierających partnerów jest zapisywana w *Monitorze Rekomendacji* (krok 13).

W zależności od typu systemu zorientowanego procesowo, rekomendowany szablon czynności może być wykorzystany na różne sposoby. Prosty scenariusz dopuszcza wyświetlenie listy czynności do wykonania każdemu członkowi VO zaangażowanemu w instancję procesu współpracy w VO. Bardziej zaawansowany scenariusz dopuszcza przekazanie szablonu czynności do elastycznego silnika przepływu prac, gdzie jest automatycznie wykonywany. Sekwencja rekomendowanych i instancjonowanych szablonów czynności, tworzona podczas wykonywania procesu doboru, może potencjalnie utworzyć pełną wykonywalną instancję procesu współpracy w VO.

Stosowanie się do rekomendacji przez członków VO biorących udział w instancji procesu współpracy w VO jest monitorowane przez *Monitor Rekomendacji* (krok 4 na Rys. 4.1). Monitoring wykorzystuje reguły zgodności (ang. *conformance rules*) wygenerowane na podstawie rekomendowanego szablonu czynności. Kolejne zdarzenia zapisywane w *Dzienniku Zdarzeń* są weryfikowane pod kątem zgodności z regułami. Spełnienie reguł zgodności, tj. stosowanie się członków VO do rekomendacji, podnosi pewność przyszłych rekomendacji. Metoda RMV obejmuje zatem pełen cykl życia szablonu czynności: odkrycie, rekomendację, instancjonowanie i monitorowanie szablonów czynności – Rys. 4.2.


Odkrycie szablonów czynności
Rekomendacja szablonów czynności
Instancjonowanie szablonów czynności
Monitorowanie szablonów czynności

Rys. 4.2. Cykl życia szablonu czynności

Podstawą rekomendacji szablonów czynności jest kontekst instancji procesu współpracy w VO, w którym znajdują się obecnie dobierający partnerzy. Szablony czynności występujące w jednej instancji procesu współpracy w VO mogą być wykorzystywane w innych instancjach VO, jeśli tylko w tych instancjach wystąpi kontekst, do którego szablony czynności pasują. Metoda RMV wspiera zatem dzielenie się wiedzą i dobrymi praktykami pomiędzy różnymi procesami współpracy w VO i różnymi wirtualnymi organizacjami.

4.3. Kluczowe elementy metody RMV

Kluczowymi elementami metody RMV są: (1) reprezentacja szablonów czynności, (2) zakres informacji w dzienniku zdarzeń procesu współpracy w VO, (3) rola usług SOVOBE w metodzie RMV, (4) model kontekstu szablonów czynności, (5) technika odkrywania i identyfikacji szablonów czynności, (6) technika generowania rekomendacji kontekstowych, (7) technika instancjonowania szablonów czynności, (8) parametryzacja metody RMV.

Reprezentacja szablonów czynności

W metodzie RMV konsekwentnie stosuje się podejście usługowe do modelowania współpracy. Każdy odkryty szablon czynności jest modelowany jako protokół usługowy. Protokół usługowy łączy w jednym modelu dwie perspektywy procesowe: perspektywę przepływu prac (model procesu, usługowe podsumowanie modelu procesu) i perspektywę społeczną (schemat sieci usługowej, sieć usługowa).

Metoda RMV jest niezależna od formalizmu wykorzystywanego do reprezentacji procesu. Użyty formalizm musi jednak pozwalać na sekwencyjne i równoległe uporządkowanie czynności. W prototypowej implementacji metody RMV do reprezentacji modelu procesu wykorzystano grafy, w których wierzchołki reprezentują czynności, a łuki – relacje czasowe między czynnościami. W rozprawie przedstawiono także przykład wykorzystania do tego celu sieci Petriego.

Zakres informacji w dzienniku zdarzeń procesu współpracy w VO

Odkrywanie szablonów czynności obejmuje: (1) odkrycie kontekstów szablonów czynności, (2) odkrycie zbioru jednostek usługowych, (3) odkrycie zbioru klas jednostek usługowych, (4) odkrycie zbioru wymagań usługowych, (5) odkrycie mapowań pomiędzy wszystkimi tymi elementami. Odkrycie szablonów czynności jest możliwe tylko wówczas, gdy każde zdarzenie w dzienniku zdarzeń jest odpowiednio opisane. Poza typowymi danymi, w metodzie RMV zakłada się, że w opisie każdego zdarzenia znajdują się następujące dane:

- *atrybuty jednostek usługowych* – niepusty zbiór atrybutów opisujących cechy jednostek usługowych zaangażowanych w wykonanie instancji czynności związanej ze zdarzeniem;
- *atrybuty społeczne* – niepusty zbiór atrybutów opisujących relacje usługowe pomiędzy jednostkami usługowymi;
- *atrybuty kontekstowe* – niepusty zbiór atrybutów opisujący kontekst, w jakim było zarejestrowane zdarzenie; zakres informacji kontekstowej jest taki sam dla każdego zarejestrowanego zdarzenia; zmiany w wartościach atrybutów kontekstowych są zapisywane w trakcie całego wykonania procesu współpracy w VO; w ten sposób atrybuty kontekstowe pozwalają na uchwycenie dynamiki kontekstu, w jakim jest wykonywana instancja procesu współpracy w VO; zakres informacji kontekstowej użyteczny w analizie danego procesu współpracy w VO w znacznym stopniu zależy od charakteru tego procesu;
- *identyfikatory jednostek usługowych* – niepusty zbiór atrybutów jednoznacznie identyfikujących jednostki usługowe zaangażowane w wykonanie instancji czynności.

Dzienniki zdarzeń zawierające taki zbiór atrybutów opisujących zdarzenie są powszechnie dostępne w nowoczesnych systemach zorientowanych procesowo. Zbiór atrybutów może być również rozszerzony o dane pochodzące ze źródeł innych niż sam system zorientowany procesowo np. portale społecznościowe. Zdarzenia w dzienniku zdarzeń procesu współpracy w VO mogą odzwierciedlać zarówno wysokopoziomowe czynności biznesowe jak i niskopoziomowe czynności wykonywane przez system informatyczny.

Rola usług SOVOBE w metodzie RMV

W metodzie RMV zakłada się, że proces współpracy w VO odbywa się między członkami SOVOBE przy wykorzystaniem usług udostępnionych przez infrastrukturę SOVOBE. Usługi SOVOBE zapewniają dostęp do danych wykorzystywanych podczas odkrywania szablonów czynności i podczas instancjonowania szablonów czynności. Informacje udostępniane każdej VO za pomocą usług SOVOBE to:

- informacje na temat kontekstu wykonania czynności,
- opisy jednostek usługowych,
- opisy relacji usługowych między jednostkami usługowymi,

- informacje o wykonywanych i wykonanych czynnościach w ramach różnych instancji procesów współpracy w VO,
- informacje o warunkach świadczenia usług.

Metoda RMV jest także jedną z usług SOVOBE udostępnianą różnym wirtualnym organizacjom. W ten sposób metoda RMV przyczynia się do współdzielenia wiedzy pomiędzy VO działającymi w ramach SOVOBE.

Model kontekstu szablonów czynności

Podstawą metody RMV jest obserwacja, że kontekst wykonania instancji czynności wpływa na sposób jej wykonania i na jej wynik, a więc pośrednio wpływa także na przebieg i wynik całej instancji procesu współpracy w VO. Kontekst instancji czynności analizowany przez metodę RMV składa się z pięciu elementów:

1. *atrybuty kontekstowe* zdarzeń związanych z instancją czynności;
2. *atrybuty instancji procesu* stanowiące statyczny opis instancji procesu współpracy w VO, wśród których jeden atrybut opisuje wynik zakończonej instancji procesu współpracy w VO;
3. *sekwencja zdarzeń*, które wystąpiły zanim wykonanie instancji czynności rozpoczęło się;
4. *aktorzy zaangażowani w realizację instancji procesu współpracy w VO,* zanim wykonanie instancji czynności rozpoczęło się;
5. *zbiór zdarzeń, które zostały zarejestrowane w innych instancjach procesu współpracy* w VO w trakcie, gdy instancja czynności była wykonywana.

Technika odkrywania i identyfikacji szablonów czynności

W metodzie RMV zachowanie współpracujących partnerów jest uznawane za szablon czynności, jeśli występuje w różnych instancjach procesu współpracy w VO predefiniowaną liczbę razy. Odkrywanie szablonów czynności w dzienniku zdarzeń obejmuje:

- odkrycie zbioru czynności i ich czasowego uporządkowania,
- odkrycie zbioru współpracujących partnerów i wykorzystywanych interfejsów usług powiązanych z czynnościami oraz cech tych partnerów i interfejsów usług oraz ich analiza w celu odkrycia klas jednostek usługowych,
- odkrycie relacji usługowych pomiędzy współpracującymi partnerami i ich analiza w celu odkrycia wymagań usługowych,
- odkrycie kontekstów szablonów czynności.

Odkrycie szablonów czynności odbywa się w dwóch krokach: (1) odkrycie wzorców sekwencji (ang. *activity sequence patterns*), (2) interpretacja wzorców sekwencji do szablonów czynności. W pierwszym kroku jest wykorzystywany zmodyfikowany algorytm *PrefixSpan* (Pei, et al., 2004), w którym wprowadzono dwie zmiany: (1) zmieniono definicję pojęć *prefiksu* i *sufiksu* w celu uchwycenia faktu wykonywania czynności w sposób równoległy; (2) odkrycie wzorców sekwencji odbywa się zarówno na poziomie czynności, jak i atrybutów czynności. W drugim kroku, interpretacja wzorców sekwencji do szablonów czynności odbywa się przez parsowanie atrybutów czynności w odkrytych wzorcach sekwencji.

Technika generowania rekomendacji kontekstowych

Analiza kontekstów szablonów czynności jej kluczowa podczas generowania rekomendacji w drugiej fazie metody RMV. Celem tej fazy jest rekomendowanie szablonów czynności, których konteksty są najbardziej podobne do kontekstu wykonywanej instancji procesu współpracy w VO.

Klasa kontekstu (ang. *context class*) jest zbiorem ograniczeń, jakie muszą być spełnione przez kontekst szablonu czynności, aby był on uznany za podobny do kontekstu instancji procesu współpracy w VO. Dopuszczalne jest, aby kontekst szablonu czynności spełniał tylko podzbiór ograniczeń z klasy kontekstu. W metodzie RMV miarą podobieństwa klasy kontekstu i danego kontekstu szablonu czynności jest *odległość kontekstu* (ang. *context distance*). Odległość kontekstu obliczana dla różnych kontekstów różnych szablonów czynności służy do wyboru najbardziej podobnego kontekstu i w konsekwencji odpowiedniego szablonu czynności.

Szablony czynności o małej odległości od kontekstu instancji procesu współpracy w VO, są dodatkowo weryfikowane przy wykorzystaniu statystycznych miar *wrażliwości* (ang. *sensitivity*) i *jednoznaczności* (ang. *specificity*) (Simon & Boring, 1990). Dla rekomendowanych szablonów czynności są obliczane: (1) *recom_index* – wskazuje oczekiwany koszt wykonania instancji procesu współpracy w VO, gdy danych szablon czynności będzie wykonany jako część instancji procesu współpracy w VO, (2) *nonrecom_index* – wskazuje oczekiwany koszt wykonania instancji procesu współpracy w VO, gdy dalsze jej wykonanie nie uwzględni szablonu czynności. Do rekomendacji są preferowane szablony czynności mające duże wartości *wrażliwości* i *jednoznaczności* oraz małą wartość *recom_index* w porównaniu z wartością *nonrecom_index*.

Rekomendowane szablony czynności są łączone w jeden *generyczny szablon czynności*, który opisuje całe zachowanie ujęte w poszczególnych rekomendowanych szablonach czynności. Częścią metody RMV jest technika łączenia szablonów czynności w jeden generyczny szablon czynności. Technika ta obejmuje łączenie modeli procesów, schematów sieci usługowych i sieci usługowych poszczególnych szablonów czynności. Końcowy wynik rekomendacji metody RMV składa się z generycznego szablonu czynności i listy szablonów czynności wraz z odległością kontekstu, wrażliwością, jednoznacznością oraz wartościami indeksów: *recom_index* i *nonrecom_index* dla każdego szablonu kontekstu.

Technika instancjonowania szablonów czynności

W najlepszym przypadku szablony czynności są odkrywane na poziomie wykonywalnym. Jeśli jednak różnorodność w wykonaniu instancji procesów współpracy w VO jest bardzo duża, to szablony czynności są odkrywane na poziomie abstrakcyjnym lub prototypowym. Takie szablony czynności mają niepełny zbiór jednostek usługowych, tzn. niektóre klasy jednostek usługowych nie mają przypisanych jednostek usługowych. Metoda RMV pozwala na efektywne wskazanie podzbioru jednostek usługowych będących członkami SOVOBE, który najlepiej pasuje do klas jednostek usługowych zdefiniowanych w szablonie czynności.

W metodzie RMV problem znajdowania jednostek usługowych dla klas jednostek usługowych jest sprowadzony do problemu poszukiwania podgrafu w sieci usługowej SOVOBE. Klasy jednostek usługowych i wymagania usługowe są wykorzystywane jako wymagania, które muszą spełnić wierzchołki i łuki podgrafu. Problem jest rozwiązany przy wykorzystaniu algorytmu genetycznego (Mitchell, 1998). W każdej iteracji algorytmu genetycznego są wykorzystywane zdefiniowane przez dobierających partnerów *funkcje zgodności*: (1) oceniające stopień dopasowania poszczególnych jednostek usługowych do odpowiednich klas jednostek usługowych, (2) oceniające poziom spełnienia wymagań usługowych przez relacje usługowe występujące pomiędzy rozważanymi w danej iteracji jednostkami usługowymi.

Parametryzacja metody RMV

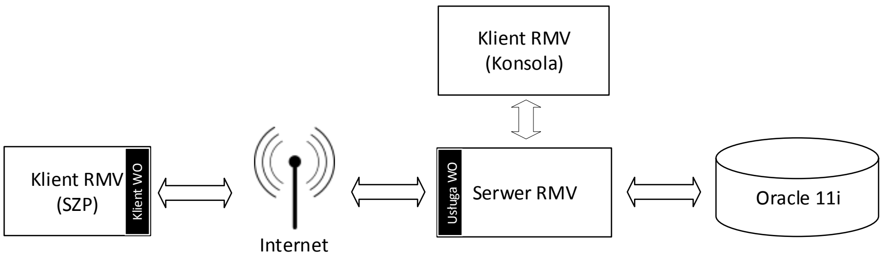
Parametryzacja metody RMV odbywa się w sposób jawny i niejawny.

Jawne parametry metody RMV są określane w chwili konfiguracji metody RMV i są przekazywane do metody RMV wraz z żądaniem rekomendacji. Parametry są wykorzystywane przez *Eksplorator Procesów* i *Menedżer Rekomendacji* (*cf.* Rys. 4.1). Wartości parametrów metody RMV muszą uwzględniać charakter analizowanego procesu współpracy w VO. Jawne parametry metody RMV w znaczący sposób wpływają na jej efektywność i jakość uzyskiwanych wyników. Parametry jawne redukują czas obliczeń wykonywanych zgodnie z metodą RMV. Lista najważniejszych jawnych parametrów metody RMV obejmuje: zakres i typ interpretowanych atrybutów zdarzeń, zakres atrybutów opisujących kontekst zdarzeń, zakres atrybutów instancji procesów w VO, minimalną liczbę atrybutów czynności w szablonach czynności, zbiór atrybutów obowiązkowych w czynnościach szablonów czynności, funkcję odległości kontekstu, funkcję mapującą konteksty instancji procesów współpracy w VO na klasy kontekstu, funkcje mapujące wartości *recom_index* i *nonrecom_index*, funkcje zgodności wykorzystywane w algorytmie genetycznym, wymagany minimalny poziom wsparcia dla szablonów czynności, wartości progowe dla funkcji zgodności wykorzystywanych w algorytmie genetycznym, liczbę szablonów czynności w pojedynczej rekomendacji, minimalną dopuszczalną wrażliwość szablonów czynności, minimalną długość szablonu czynności.

Niejawne preferencje partnerów dobierających są określane przez *Monitor Rekomendacji* na podstawie analizy wyborów dokonywanych przez dobierających partnerów i skłonności członków VO do podążania za rekomendacją. Dla każdego szablonu czynności *Monitor Rekomendacji* oblicza *wskaźnik pewności* (ang. *confidence indicator*), a dla każdej instancji procesu współpracy w VO – *współczynnik społeczny* (ang. *social coefficient*). Wskaźnik pewności danego szablonu czynności określa chęć członków VO do podążania za rekomendacją. Współczynnik społeczny wskazuje na preferencje członków VO do podążania za szablonami czynności odkrytymi w instancjach procesu współpracy w VO, w których sami brali udział lub wręcz przeciwnie. Wartości parametrów niejawnych wpływają na uporządkowanie szablonów czynności na liście rekomendacji.

5. Prototyp i ewaluacja metody RMV

Prototyp metody RMV został zaimplementowany w architekturze klient-serwer (Rys. 5.1) w języku Java^{TM}. Logika metody RMV znajduje się na *serwerze RMV*. Warstwa danych została zrealizowana przy wykorzystaniu systemu baz danych Oracle 11i. Prototyp ma dwa klienty: ograniczona funkcjonalność serwera jest dostępna za pomocą *klienta konsolowego* (odkrywanie wzorców sekwencji, identyfikacja szablonów czynności), pełna funkcjonalność wymaga wykorzystania *systemu zorientowanego procesowo*.

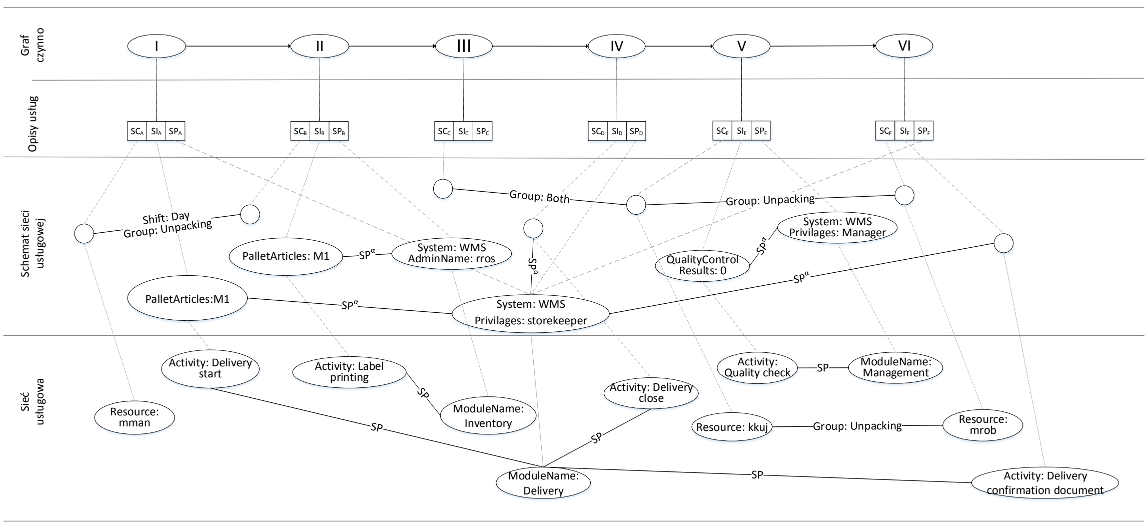


Rys. 5.1. Architektura prototypu metody RMV

Przykładem systemu zorientowanego procesowo, z którym został zintegrowany prototyp metody RMV, jest system *ErGo*. System *ErGo* (http://ergo.kti.ue.poznan.pl/) (Paszkiewicz, et al., 2011) jest wykorzystywany do wsparcia wykonania procesów budowalnych zarządzanych przez deweloperów nieruchomości.

Jakość rekomendacji generowanych przy wykorzystaniu metody RMV została zweryfikowana podczas analizy rzeczywistych danych firmy Epsilon. Przeprowadzona dla firmy Epsilon analiza miała na celu udzielenie rekomendacji dotyczących organizacji pracy magazynierów zaangażowanych w realizację procesów magazynowych tej firmy. Przedmiotem analizy był proces obsługi palet z materiałami produkcyjnymi, który jest wspierany przez system zarządzania magazynem. System zarządzania magazynem jest systemem zorientowanym procesowo. Wewnętrzna organizacja firmy Epsilon i cechy procesu dostawy materiałów produkcyjnych pokrywają się z cechami wirtualnej organizacji i cechami procesu współpracy w VO.

Proces obsługi palet z materiałami produkcyjnymi obejmuje czynności związane z przyjęciem materiałów do magazynu, przechowywaniem palet w magazynie i wydaniem materiałów do produkcji. Eksperymenty prowadzone na danych firmy Epsilon pozwoliły na jakościową i ilościową ocenę rekomendacji generowanych przez metodę RMV. Przykładowy szablon czynności odkryty w dzienniku zdarzeń systemu zarządzania magazynem jest przedstawiony na Rys. 5.2.



Rys. 5.2. Szablon czynności odkryty dla procesu zarządzania materiałami produkcyjnymi w firmie Epsilon

Szablon czynności na Rys. 5.2 składa się z sześciu czynności *I-VI*. Każda czynność jest powiązana z jednym opisem usługi. Elementy poszczególnych opisów usług są powiązane z klasami jednostek usługowych pochodzących ze schematu sieci społecznej. Nie wszystkie elementy opisów usług są powiązane z klasami jednostek usługowych. Część klas jednostek usługowych jest powiązana z jednostkami usługowymi z sieci usługowej. Szablon czynności z Rys. 5.2 pozwolił na zdefiniowanie następujących rekomendacji: jeśli celem biznesowym, jest zmniejszenie całkowitego czasu wykonania instancji procesu, a wykonanie instancji procesu wiąże się z materiałami typu *M1*, wykonanie instancji procesu powinno rozpocząć się od wywołania usługi *Delivery start* systemu magazynowego przez pracownika *mman*. Druga czynność powinna uwzględniać wywołanie usługi *Label printing* modułu *Inwentory* systemu magazynowego. Czynność *III* powinna być wykonana przez magazyniera pochodzącego z grupy *Both*, do której należy także magazynier *kkuj* zaangażowany w wykonanie czynności *V*. Czynność *IV* powinna być wykonana przy użyciu usługi *Delivery close* oferowanej przez moduł *Delivery* systemu magazynowego. Dane związane z tą usługą powinny być zapisane do bazy danych z uprawnieniami roli *Storekeeper*. Wykonanie czynności *V* powinno wiązać się z wywołaniem przez magazyniera *kkuj* usługi *Quality check* modułu *Management* systemu magazynowego. *Kkuj* musi należeć do grupy *Unpacking* razem z magazynierem *mrob* zaangażowanym w wykonanie czynności *VI*. Czynność *VI* powinna być wykonana przez wywołanie usługi *Delivery confirmation document* oferowanej przez nieokreślony w szablonie czynności moduł systemu magazynowego.

Przedstawiona powyżej rekomendacja jest bardzo szczegółowa. W praktyce, w celu ułatwienia zrozumienie wygenerowanych rekomendacji były one uogólniane przed przekazaniem firmie Epsilon. Rekomendacje pozwoliły firmie Epsilon na identyfikację magazynierów, grup magazynierów i zmian magazynowych, które mają albo pozytywny, albo negatywny wpływ na funkcjonowanie magazynu. Rekomendacje doprowadziły do szeregu działań podjętych przez firmę Epsilon, takich jak szkolenia pracowników, zmiana przypisania pracowników do zmian, modyfikacja obciążenia zmian magazynowych czy rekonfiguracja systemu magazynowego.

6. Wnioski

Przedstawiona w rozprawie metoda rekomendacji dla wirtualnych organizacji RMV stanowi rozwiązanie problemu efektywnego wsparcia informatycznego dla nieprzewidywalnych, wyłaniających się i nieustrukturyzowanych procesów współpracy w VO. Wsparcie dla wykonania procesów współpracy w VO ma formę kontekstowych rekomendacji szablonów czynności.

Idea metody RMV jest oparta na czterech spostrzeżeniach:

1. dziennik zdarzeń procesu współpracy w VO zawiera informacje opisujące interakcje zachodzące pomiędzy aktorami współpracującymi w ramach poszczególnych instancji procesu współpracy w VO;
2. kontekst, w którym odbywa się współpraca, ma znaczący wpływ na kształt tej współpracy;
3. powtarzające się zachowania współpracujących aktorów, nazywane szablonami czynności, mogą być odkryte na podstawie analiz dzienników zdarzeń;
4. odkryte szablony czynności mogą zostać ocenione jako warte lub nie warte rekomendacji; rekomendowane szablony czynności przyczyniają się podniesienia efektywności wykonania instancji procesów współpracy w VO.

Metoda RMV składa się z czterech części: (1) odkrywanie sekwencji czynności, (2) identyfikacja szablonów czynności, (3) formułowanie rekomendacji, (4) instancjonowanie szablonów czynności. Każda z tych części jest źródłem wartości dodanej sama w sobie, ale wykonane sekwencyjnie pozwalają na efektywne wsparcie procesów współpracy w VO spełniające postawione w dysertacji wymagania dla takiego wsparcia (*cf.* Rozdział 4.1). współpracy w WO. Metoda RMV maksymalizuje wsparcie dla użytkownika w postaci rekomendacji szablonów czynności odpowiednich do wykonania w bieżącym kontekście procesu współpracy w VO (wymaganie 1). W przypadku zmiany kontekstu instancji procesu współpracy w VO, zbiór rekomendowanych czynności jest inny (wymaganie 3 i 8). Decyzja dotycząca ostatecznego wyboru i instancjonowania szablonu czynności jest podejmowana przez grupę dobierających partnerów, a nie jest dokonywana automatycznie (wymaganie 5). Szablony czynności zawierają nie tylko specyfikację czynności, ale także specyfikację cech i relacji usługowych pomiędzy członkami VO (wymaganie 8). Rekomendowane szablony czynności są modelami deskryptywnymi a nie preskryptywnymi (wymaganie 4). Szablony czynności odkryte w procesach współpracy jednej VO mogą być wykorzystywane wielokrotnie w procesach współpracy innych VO (wymaganie 6 i 7). Mechanizm monitorowania rekomendacji jest podstawą oceny gotowości członków VO do podążania za rekomendacją (wymaganie 2). Wreszcie metoda RMV wspiera ciągłe instancjonowanie procesu współpracy w VO. Dobór nowych partnerów i usług jest wykonywany za każdym razem, gdy dobierający partnerzy akceptują do wykonania rekomendowany szablon czynności (wymaganie 9). Dobór partnerów i usług odbywa się w oparciu o kryteria społeczne, istotne dla procesów współpracy w VO (wymaganie 8).

Metoda RMV została wykorzystana w praktyce do analizy dzienników zdarzeń pochodzących z systemu zarządzania magazynem firmy produkcyjnej. Nietrywialne i trafne rekomendacje otrzymane w wyniku zastosowanie metody RMV miały dużą wartość biznesową dla danego procesu współpracy w VO. Klasy jednostek usługowych i wymagania usługowe stanowiące część szablonów czynności były źródłem wartościowej informacji o czynnikach sukcesu procesu współpracy w VO. Wiedza na temat czynników sukcesu została wykorzystana do doboru jednostek usługowych, które są w stanie wykonać proces współpracy w VO w bardziej efektywny sposób.

Główne osiągnięcia rozprawy obejmują:

- identyfikację i ocenę opisanych w literaturze metod doboru partnerów i usług w dziedzinie sieci współpracy i architektury usługowej pod kątem zastosowania w instancjonowaniu procesów współpracy w VO;
- identyfikację i ocenę istniejących metod rekomendacji czynności w dziedzinie systemów zorientowanych procesowo, kontekstowych systemów rekomendacyjnych i eksploracji procesów pod kątem wykorzystania w informatycznym wsparciu wykonania procesów współpracy w VO;
- model formalny procesu współpracy w VO, szablonu czynności, kontekstu szablonu czynności, dziennika zdarzeń procesu współpracy w VO;
- opracowanie metody odkrywania i identyfikacji szablonów czynności, która pozwala na wyekstrahowanie szablonów czynności i ich kontekstów z dziennika zdarzeń procesu współpracy w VO tworzonego przez system zorientowany procesowo;
- opracowanie metody formułowania rekomendacji szablonów czynności w wykonywanych procesach współpracy w VO, gdzie dopasowanie szablonu czynności do instancji procesu współpracy w VO opiera się na analizie aktualnego kontekstu instancji i kontekstów szablonów czynności;
- opracowanie metody instancjonowania abstrakcyjnych i prototypowych szablonów czynności, gdzie instancjonowanie odbywa się w ramach SOVOBE i ma miejsce cały czas podczas wykonania instancji procesu współpracy w VO;
- implementacja prototypu metody RMV i integracja prototypu z systemem zorientowanym procesowo *ErGo* wykorzystywanym w sektorze budowalnym do wsparcia wykonania procesu budowlanego;
- wykorzystanie metody RMV do analizy dzienników zdarzeń firmy produkcyjnej; przeprowadzona analiza doprowadziła do nietrywialnych rekomendacji, ocenionych jako bardzo wartościowe przez kierownika magazynu i dystrybucji firmy Epsilon.

Dwiema ważnymi cechami metody RMV jest jej rozszerzalność i niezależność. *Rozszerzalność* metody RMV polega na elastycznej definicji zbioru atrybutów i funkcji wykorzystywanych podczas odkrywania szablonów czynności i ich rekomendacji. Różne zbiory atrybutów, istotne dla danego obszaru aplikacji, mogą składać się na opisy jednostek usługowych, relacji usługowych i kontekstu. Skomplikowanie wykorzystywanych funkcji zależy od użytkownika metody RMV. W zależności od potrzeby, funkcje mogą pozwalać na bardzo wyrafinowane i dogłębne analizy. Metoda RMV jest niezależna od typu analizowanego procesu współpracy w VO i systemu zorientowanego procesowo. Metoda RMV może być zastosowana do analizy każdego dziennika zdarzeń, który ma cechy dziennika zdarzeń procesu współpracy w VO. Niezależność metody RMV pozwala na jej zastosowanie w różnych procesach współpracy w VO i różnych domenach biznesowych. Poza zastosowaniem do analizy procesów magazynowych, które przedstawiono w rozprawie, metoda RMV jest obecnie używana do analizy procesów obiegu dokumentów w Wielkopolskim Urzędzie Wojewódzkim w Poznaniu.

Metoda RMV była prezentowana m.in. podczas seminarium *„Unleashing Operational Process Mining”*[19] w Daghstul “ zorganizowanymi przez Grupę Roboczą IEEE ds. Eksploracji Procesów[20], podczas *World Business Congress* zorganizowanego przez międzynarodową organizację *International Management Development Association* (Paszkiewicz & Cellary, 2011), podczas dwóch konferencji IFIP poświęconej tematyce sieci współpracy (Paszkiewicz & Picard, 2010) (Paszkiewicz & Picard, 2009), podczas konferencji poświęconej teorii i praktyce elektronicznej administracji ICEGOV (Paszkiewicz & Cellary, 2012), podczas piętnastej konferencji poświęconej systemom współpracy CSCWD (Paszkiewicz & Picard, 2011) i podczas dwóch sympozjów doktoranckich organizowanych przy okazji konferencji BIS 2013[21] i ADBIS 2012[22]. Metoda RMV jest także opisana w czasopiśmie *Journal of Transnational Management* (Paszkiewicz & Cellary, 2012). Elementy metody znalazły się w dwóch rozdziałach książek (Picard, et al., 2014) (Picard, et al., 2010). Wartość analiz sieci społecznej w analizie procesów biznesowych została przedstawiona w (Paszkiewicz & Picard, 2013)

Metoda RMV otwiera nowe kierunki badań. Interesująca jest możliwość analizy wzajemnego wpływu na siebie struktury czynności wchodzących w skład modelu procesu i struktury sieci społecznych. Jest to możliwe, ponieważ szablony czynności łączą dwie perspektywy procesowe: perspektywę przepływu sterowania i perspektywę społeczną. Analiza wzajemnego wpływu tych dwóch perspektyw na siebie jest nową, ciekawą i obiecującą dziedziną badań. Badania takie wymagają rozszerzenia metody RMV. Takie rozszerzenie musi objąć symulację i predykcję wpływu zmian wprowadzanych w jednej perspektywie na zmiany w strukturze i cechach drugiej perspektywy. Przykładem analizowanej cechy sieci społecznej może być odporność sieci na uszkodzenia. Przykładowe cechy perspektywy przepływu sterowania obejmują efektywność wykonania zadań i strukturę zależności pomiędzy zadaniami. Rozszerzona metoda RMV byłaby ważnym krokiem naprzód w rozwoju metod analizy dynamiki pracy zespołów, tworzenia zespołów i grup organizacji z potencjalnymi obszarami aplikacji takimi jak inteligentne miasta (ang. *smart cities*) lub sektor budowlany.

[19] “Unleashing Operational Process Mining”, seminarium Dagstuhl https://www.dagstuhl.de/en/program/calendar/semhp/?semnr=13481

[20] Grupa Robocza IEEE ds. Eksploracji Procesów, www.win.tue.nl/ieeetfpm/

[21] Sympozjum doktoranckie organizowane przy okazji 16th International Conference on Business Information Systems (BIS 2013), http://bis.kie.ue.poznan.pl/16th_bis/phd2013.php

[22] Sympozjum doktoranckie organizowane przy okazji 16th East-European Conference in Advances in Databases and Information Systems, http://adbis.cs.put.poznan.pl/call_phd_consortium.php

Bibliografia do streszczenia polskiego

Aalst, W. M. P., 2004. Business Process Management: A Personal View. *Business Process Management Journal,* II(10).

Aalst, W., 2009. *TomTom for Business Process Management (TomTom4BPM).* Amsterdam, The Netherlands, Springer-Verlag, p. 2–5.

Aalst, W., 2011. *Process Mining. Discovery, Conformance and Enhancement of Business Processes.* : Springer.

Aalst, W., Pesic, M. & Song, M., 2010. Beyond process mining - from the past to present and future. *Proceedings of the 22nd international conference on advanced information systems engineering,* pp. 38-52.

Aalst, W., Schonenberg, M. & Song, M., 2011. Time Prediction Based on Process Mining. *Information Systems,* II(36), p. 450–475.

Abowd, G. et al., 1999. *Towards a Better Understanding of Context and Context-Awareness.* London, UK, Springer-Verlag, pp. 304-307.

Adomavicius, G., Sankaranarayanan, R., Sen, S. & Tuzhilin, A., 2005. Incorporating Contextual Information in Recommender Systems Using a Multidimensional Approach. *ACM Transactions on Information Systems,* I(23), pp. 103-145.

Adomavicius, G. & Tuzhilin, A., 2005. Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering,* XVII(6), pp. 734-749.

Agrawal, R. & Srikant, R., 1995. *Mining sequential patterns.* Washington, DC, USA, EEE Computer Society, pp. 3-14.

Brendel, R. & Krawczyk, H., 2010. *Static and Dynamic Approach of Social Roles Identification Using PISNA and Subgraphs Matching.* Taiyuan, IEEE Computer Society, pp. 557 - 560.

Camarinha-Matos, L. et al., 2007. *A Computer-Assisted VO Creation Framework.* Guimarães, Portugal, Springer, pp. 165-178.

Camarinha-Matos, L., Afsarmanesh, H. & Ollus, M., 2008. ECOLEAD and CNO Base Concepts. *Methods and Tools for Collaborative Networked Organizations,* p. 3–32.

Canfora, G., Di Penta, M., Esposito, R. & Villani, M., 2005. *An approach for QoS-aware service composition based on genetic algorithms.* New York, NY, USA, ACM, pp. 1069-1075.

Cellary, W. & Strykowski, S., 2009. *E-government Based on Cloud Computing and Service-Oriented Architecture.* Bogota (Colombia), ACM Press, p. 5–10.

Cellary, W. & Strykowski, S., 2009. *E-government Based on Cloud Computing and Service-Oriented Architecture.* Bogota (Colombia), ACM Press, p. 5–10.

Claro, D., Albers, P. & Hao, J., 2005. *Selecting Web Services for Optimal Composition.* Orlando, USA, Springer.

Crispim, J. & Sousa, J., 2007. *Multiple Criteria Partner Selection in Virtual enterprises.* Guimarães, Portugal, Springer, pp. 197-206.

Delias, P. et al., 2013. *Clustering Healthcare Processes with a Robust Approach.* Rome, Italy, EURO-INFORMS.

Dey, A., 2001. Understanding and Using Context. *Personal and Ubiquitous Computing,* 5(1), pp. 4-7.

Ding, H., Benyoucef, L. & Xie, X., 2003. *A Simulation-Optimization Approach Using Genetic Search for Supplier Selection.* New Orleans, Louisiana, USA, IEEE Computer Society.

Dorn, C., Burkhart, T., Werth, D. & Dustdar, S., 2010. *Self-Adjusting Recommendations for People-driven Ad-hoc Processes.* Hoboken, NJ, USA, Springer-Verlag Berlin, Heidelberg, pp. 327-342.

Dumas, M., Aalst, W. & Hofstede, A. H., 2005. *Process-Aware Information Systems: Bridging People and Software Through Process Technology.* Hoboken, NJ, USA: John Wiley & Sons, Inc..

Ermilova, E. & Afsarmanesh, H., 2010. Competency Modeling Targeted on Boosting Configuration of Virtual Organizations. *Production Planning and Control. The Management of Operations*, II(21), pp. 103-118.

Gallon, M., Stillman, H. & Coates, D., 1995. Putting Core Competency Thinking Into Practice. *Research Technology Management,* III(38), pp. 20-29.

Haisjackl, C. & Weber, B., 2011. *User Assistance During Process Execution - an Experimental Evaluation of Recommendation Strategies.* Hoboken, New Jersey, USA, Springer, pp. 135-145.

Hwang, S., Wei, C. & Yang, W., 2004. Discovery of temporal patterns from process instances. *Computers in Industry - Special issue: Process/workflow mining,* 53(3), pp. 345 - 364.

Jaeger, M. & Mühl, G., 2007. *QoS-based Selection of Services: The Implementation of a Genetic Algorithm.* Bern, Switzland, IEEE Computer Society.

Mane, R., 2013. A comparative study of Spam and PrefixSpan sequential pattern mining algorithm for protein sequences. *Advances in Computing, Communication, and Control Communications in Computer and Information Science,* Volume 361, pp. 147-155.

Mitchell, M., 1998. *An Introduction to Genetic Algorithms.* Cambridge, MA, USA: MIT Press.

Morzy, M. & Forenc, K., 2013. Social Network Analysis on Highly Aggregated Data: What Can We Find?. In: T. Morzy, T. Härder & R. Wrembel, eds. *Advances in Databases and Information Systems.* Germany: Springer, pp. 195-206.

Nakatumba, J., Westergaard, M. & Aalst, W., 2012. *A meta-model for operational support. BPM Center Report.* [On-line]
Available at: http://bpmcenter.org/wp-content/uploads/reports/2012/BPM-12-05.pdf
[Accessed 26 May 2012].

OASIS Technical Committee, 2006. *Reference Model for Service Oriented Architecture 1.0. OASIS Standard..* [On-line]
Available at: https://www.oasis-open.org/committees/download.php/19679/
[Accessed 17 5 2013].

Paszkiewicz, Z., Gabryszak, P., Krysztofiak, K., Wawrzyniak, K., Picard, W., 2011. *ErGo: Developer's Guide,* Poznań: Poznań University of Economics, Department of Information Technology.

Paszkiewicz, Z. & Cellary, W., 2011. *Computer supported collaborative processes in virtual organizations.* Poznań, IMDA Press, pp. 85-94.

Paszkiewicz, Z. & Cellary, W., 2012. *Computer supported contractor selection for public administration ventures.* Albany, NY, ACM, pp. 322-335.

Paszkiewicz, Z. & Cellary, W., 2012. Computer supported collaboration of SMEs in transnational market. *Journal of Transnational Management,* 17(4), pp. 294-313.

Paszkiewicz, Z. & Picard, W., 2009. *Modeling virtual organization architecture with the Virtual Organization Breeding Methodology.* Thessaloniki, Greece, Springer, pp. 187-196.

Paszkiewicz, Z. & Picard, W., 2010. *MAPSS, a Multi-Aspect Partner and Service Selection method.* Saint-Etienne, France, Springer, pp. 329-337.

Paszkiewicz, Z. & Picard, W., 2011. *Modeling competences in service-oriented virtual organization breeding environments.* Lausanne, Switzerland, IEEE, pp. 497-502.

Paszkiewicz, Z. & Picard. W., 2013. *Analysis of the Volvo IT Incident and Problem Handling Processes using Process Mining and Social Network Analysis.* Beijing, China, CEUR online proceedings.

Pei, J. et al., 2004. Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering,* Volume 16.

Picard, W., 2013. A Formalization of Social Requirements for Human Interactions with Service Protocols. *Information Sciences,* Volume 283, pp. 1-21.

Picard, W., 2013. *Adaptation of Service Protocols.* Poznań, Poland: Poznań University of Economics Press.

Picard, W. et al., 2010. Breeding Virtual Organizations in a Service-Oriented Architecture Environment. In: *SOA Infrastructure Tools - Concepts and Methods.* Poznań, Poland: Poznań University of Economics Press, p. 375–396.

Picard, W. & Cellary, W., 2010. *Agile and Pro-active Public Administration as a Collaborative Networked Organization.* New York (NY, USA), ACM, pp. 9-14.

Porter, M., 2008. *Competitive Advantage: Creating and Sustaining Superior Performance.* New York, USA: Simon and Schuster.

Rabelo, R. & Gusmeroli, S., 2008. *The ECOLEAD Collaborative Business Infrastructure for Networked Organizations.* Poznań, Springer, p. 451–462.

Reichert, M., 2011. *What BPM Technology Can Do for Healthcare Process Support.* Springer-Verlag, s.n., p. 2–13.

Ricci, F., Rokach, L., Sshapira, B. & Kantor, P. B., 2011. *Recommender Systems Handbook.* : Springer.

Simon, D. & Boring, J., 1990. Sensitivity, Specificity, and Predictive Value. In: H. Walker, W. Hall & J. Hurst, eds. *Clinical Methods: The History, Physical, and Laboratory Examinations.* Boston: Butterworths, p. Chapter 6 .

Sinur, J. & Jones, T., 2012. *Leverage Automated Business Process Discovery for Business Benefits,* Stamford, CT, USA: Gartner Report.

Srikant, R. & Agrawal, R., 1996. *Mining sequential patterns: generalizations and performance improvements.* London, UK, Springer-Verlag

Stoner, J., Freeman, R. & Gilbert, D. R., 1999. *Management.* 6 ed. Singapore: Pearson.

Swinkels, G., 2012. *Performance Improvement based on Cross-Organizational Recommendations,* Eindhoven : Eindhoven University of Technology.

Tan, P., Goh, A. & Lee, S., 2008. *A Context Model for B2B Collaborations.* Washington, DC, USA, IEEE Computer Society, pp. 108-115.

Wang, J. & Han, J., 2004. *BIDE: efficient mining of frequent closed sequences.* Boston, MA, USA, IEEE Computer Society, pp. 79 - 90.

Watts, D., 2004. *Six Degrees: the Science of a Connected Age.* New York, NY, USA: W. W. Norton & Company.

Witten, I., Frank, E. & Hall, M., 2011. *Data Mining. Practical Machine Learning Tools and Techniques.* Third ed. Burlington, MA, USA: Elsevier Inc..

Workflow Management Coalition, 1999. *Terminology and Glossary.* [On-line] Available at: http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf [Accessed 19 October 2012].

Podziękowanie

Praca nad rozprawą została sfinansowana ze środków Narodowego Centrum Nauki przyznanych na podstawie decyzji nr DEC-2011/01/N/ST6/04205