Rule-Based Service Modeling

Michael Gebhart, Sebastian Abeck
Research Group Cooperation & Management
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
{gebhart | abeck} @cm-tm.uka.de

***Abstract*— In the context of service-oriented computing, services provide the capabilities necessary to support the business, especially its processes. Within the service modeling process, the services that constitute a common and stable basis for all supported processes have to be identified. The consideration of each business process on its own, without guidelines, may yield a service inventory that is not adjusted with all requirements, and for instance contains several services providing the same capability, having an inappropriate granularity, or using different taxonomies. Thus, this paper suggests applying rules on a model capturing relevant business requirements to systematically derive a blueprint as a proposal of necessary services. The approach is exemplified by a scenario at the Karlsruhe Institute of Technology (KIT), where it is applied in the context of module catalog management.**

Keywords-service modeling, service inventory blueprint, service candidates, rules

I. INTRODUCTION

In the context of service-oriented computing, services as building blocks [14] provide the capabilities necessary to support the business. They exist as physically independent software programs and have assigned their own distinct functional contexts.

The establishment of a service-orientation requires the identification of the necessary services and their capabilities [3, 15]. Before these services can be implemented, a service inventory blueprint describing the potential services and their capabilities in a conceptual manner is required. The services within this blueprint are called service candidates, and they group the capabilities related to the context of the service candidate. The capabilities are called service capability candidates. The blueprint is created during the service modeling process as part of the service-oriented analysis process [1]. It is required to provide a common and stable basis that is at best adjusted to all supported processes.

To determine service candidates, a typical iterative top-down approach is to regard every business process that has to be supported on its own [1]. The business process is decomposed until service capability candidates for specific activities can be derived. Existing service candidates are revised or new service candidates are added to the blueprint. When applying this approach, the question is: When should the decomposition be stopped to obtain reusable but also not too fine-grained services? Additionally, regarding one business process after the other may inhibit the creation of a common set of service candidates, for the relationship between single activities of decomposed business processes is not formalized. When deriving service candidates from the decomposed business processes, a common ground between the processes is not identified and process "silos" of services are produced [4]. The service capabilities and their design, such as their granularity, are aligned to a specific business process. This means that they are designed to exactly fulfill the requirements necessary for one specific business process. Maybe existing service candidates are not regarded since their granularity is inappropriate or the taxonomy differs. Instead, new services are created. This may result in several services providing the same capability with different granularities or names. A continuous revision of the service inventory blueprint is required to avoid these issues, which constitutes a complex task. The taxonomy has to be unified and a compromise to satisfy all requirements has to be found. This requires all prior design decision to be kept in mind.

The contribution of this paper is a set of rules to systematically derive a service inventory blueprint from a customized business model capturing relevant requirements in one model. The customized business model focuses the relevant information for service candidate derivation and primarily describes "what" is done within the organization. The information regarding "how" things are done is described by business processes and is not required for service derivation. Thus, the business model includes the business performer roles, business activities and affected business entities. The description of these aspects within one model enables the unification of taxonomies, levels of abstraction at one place and the automatic and tool-supported application of formalized rules. Additionally, it helps to identify redundant activities that would result in redundant services later. An iterative or subsequent revision of the service inventory blueprint, including steps as service normalization [12] that expect all design decisions to be kept in mind, is not required.

Our approach is exemplified by a scenario at the Karlsruhe Institute of Technology (KIT). In the context of teaching, modules describe the teaching content of degree courses and are listed within a module catalog that has to be managed. The entire module catalog management is expected to be supported by IT, in particular by services. At the KIT, we describe the module catalog management by

means of the customized business model and use business processes, standardized documents describing the module catalog management and interviews as input. Afterwards, our rules are applied to systematically identify a common and stable set of service candidates to support the module catalog management and its business processes.

The paper is organized as follows: Section 2 presents the related work in the context of service inventory blueprints, business modeling and derivation of services. In Section 3, the customized business model including its meta model and the rules to derive the service inventory blueprint are described. Additionally, this section shows the integration of the service inventory blueprint and its derivation into an entire SOA development process. This illustrates the importance of the blueprint and its advantages in combination with various existing approaches that start with business process models. The approach is exemplified by the prior described scenario at the Karlsruhe Institute of Technology (KIT). Section 4 concludes the paper and makes suggestions for future research work.

II. Related Work

In [1] the concept of a service inventory blueprint is introduced as a conceptual blueprint of all planned services, called service candidates. Each service candidate contains possible capabilities, called service capability candidates. The availability of this blueprint dramatically reduces the effort and risk associated with the design of reusable services. To derive the service inventory blueprint in [1] the business processes are considered and the blueprint is continuously revised. We advocate the idea of a blueprint but see the need for a systematic approach to derive the blueprint to avoid continuous revision.

An approach to categorize services is introduced in [1, 13]. Here, a distinction of entity services, task services and utility services is proposed. Entity services focus on one business entity, while task services have process logic that does not fit within a functional context of one business entity. Additionally, entity services are process agnostic, whereas task services are non-agnostic. This means that task services are only specific for one business process. Utility services provide infrastructure functionality. Thus, they do not affect business entities. A similar categorization can be found in [2]. Our approach uses this distinction to encapsulate identified capabilities into service candidates applied in rule 5 to rule 8.

According to [4] "organizations are structured around their key functions and the end-to-end process goes between those functions". Process-based service discovery tends to "drill-down" too early and to produce process "silos" of services. This often fails to identify common ground between processes. Thus, service discovery should start with the key functions describing the "what". We agree with that, but miss any formalized method to describe the business and to systematically derive the services.

Another approach using the relation of activities to business entities is introduced in [3]. The resulting services correspond to both entity services and task services. They are not further distinguished. Considering the relation between activities and business entities is a very promising method, which is why it is reused in our approach. However, we see the need to describe the business within one model to first find redundant activities and second to unify the taxonomy. Additionally, the method should be applicable directly on the created business model. The approach in [3] could be further improved if dependencies between business activities and potential compositions of business entities were described. This enables the identification of service relevant business activities and business entities, for not every activity has to be supported as a service and other activities should be explicitly supported to increase the reusability.

Several approaches combined can be found in [11]. Here, also entity task and utility services are distinguished as introduced in [1]. The approach could be improved with a systematic and formalized guideline for how to identify service relevant business activities and the service candidates that are currently presumed.

As enhancement of the classic RUP [10], in [6] the Rational Unified Process (RUP) for Service-Oriented Modeling and Architecture [9] is introduced. Here, a method for business modeling which can be applied using the Unified Modeling Language (UML) or an UML profile [7, 8] is proposed. However, it does not exactly fulfill the requirements for service modeling. For service modeling, the activities within an organization, the relation to business processes, the performing role and the affected business entities have to be described. Additionally, the relation between activities and the composition of business entities has to be modeled. Thus, we decided to create a customized business model that exactly meets these requirements. However, our design decisions reuse concepts such as business use cases and their realization in the form of business processes as introduced in [6, 7, 8, 9].

III. Rule-based Service Modeling

This section introduces the customized business model specific for service modeling and the rules to derive a service inventory blueprint. In a first step, the meta model of the business model is described. Afterwards, the rules are presented to derive the service inventory blueprint. To exemplify our approach, a scenario at the KIT in the context of module catalog management is used. The resulting service candidates of the service inventory blueprint are presented. Afterwards, formalizations for two selected derivation rules are shown, using the Object Constraint Language (OCL) [18]. This exemplifies the systematic and automatic application of the presented derivation rules. In a last step, the service inventory blueprint and its derivation are embedded into an entire SOA development process to illustrate the combination with existing approaches.

A. Business Meta Model

In this section, the business model and its meta model are introduced. The business meta model is customized to the specific needs for a systematic service candidate derivation. It uses elements and the taxonomy according to the IBM business modeling [6, 7, 8], but simplified and focused on the aspects required to derive service candidates.

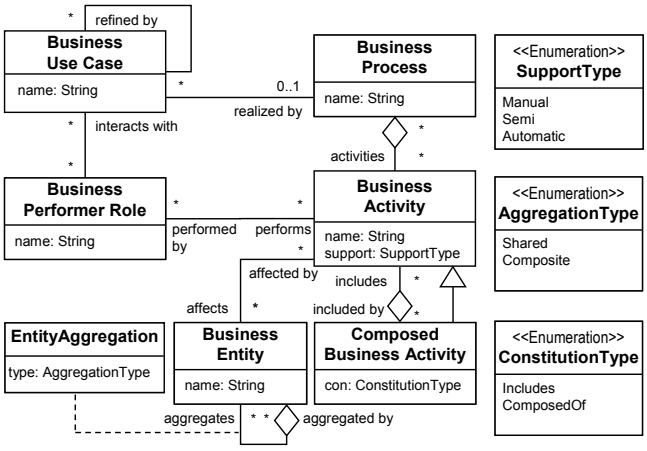

Figure 1. Business Meta Model.

Additionally, some aspects of the UML superstructure as shared and composite aggregation [16] and the Business Process Definition MetaModel (BPDM) 1.0 standard as the business performer role [17] are included. Figure 1 illustrates the business meta model.

To derive service candidates, it is necessary to identify what is done within an organization [4]. This is formalized by a top-level business use case interacting with several external business actors [7] represented as business performer roles. The business use case can be refined into more detailed business use cases. Recommended are two levels of business use cases [6]: The first level describes the goals of the organization. The second level illustrates the internal business processes from an external view [6, 7]. Each of these business use cases can be realized by a business process. In contrast to business use cases, the business processes describe how internal business workers interact and which information they use to realize the behavior described in the business use cases. The distinction of business workers and business actors is not necessary. Thus, both are represented as a business performer role. Since business activities describe what is done within an organization, they are required to identify service capability candidates. Activities can be part of composed activities or business processes. A composed activity is either completely composed of the subordinate activities or it only includes the subordinate activities and adds some individual functionality. Additionally, for each activity, the IT support can be set by the support attribute. It can be automatic, semi or manual. Manual means that there is no IT support and automatic means full IT support. Semi represents a partial IT support, for instance for business activities that are composed of a manual and an automatic business activity. The composition mechanism of activities enables the determination of the reuse of functionality. Each business activity can be invoked by a business performer role [17]. This covers explicitly invoked activities only, not implicitly invoked activities. An explicitly invoked activity is required as service, anyway. To distinguish between different service categories, such as entity and task services [1, 2, 13] and to group service capability candidates to service candidates, the relation to business entities has to be considered [1]. A business activity may affect no, one or more than one business entity. Business entities can be comprised of other entities. Here the type can be set to "shared" or "composite" as defined in the UML superstructure for aggregations [16]. When using the composite aggregation, the subordinate business entity is not able to exist on its own.

For the module catalog management scenario, the following business use cases and business processes are identified. The business use case "Module Catalog Management" can be refined by the business use cases "Update Module Catalog" and "Create Module Catalog". These use cases are realized by similar named business processes, as illustrated in Figure 2.

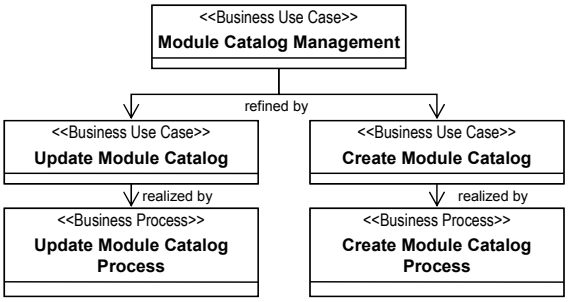

Figure 2. Module Catalog Management Business Use Case.

B. *Service Inventory Blueprint Creation*

In the following, eight rules are described to systematically derive the service candidates and their capabilities. The first four rules are used to identify the business activities that represent service capability candidates. These capability candidates have to be grouped into service candidates. For this purpose, the latter four rules are used.

A business activity represents a service capability candidate if one of the following rules applies:

1) The business activity is IT supported (automatic) and is associated with at least one performer role.

Explanation: The capability is explicitly called and thus explicitly required as capability.

Figure 3 shows an excerpt of the module catalog management business model where this rule can be applied. The business activity "Update Module" is directly called by the performer role "Module Catalog Manager" and thus identified as service capability candidate. For the performer role "Module Catalog Manager" the notation for business workers as proposed in [8] is used, for this performer role is internal to the organization.

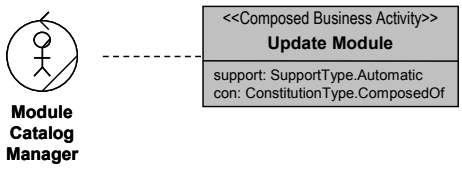

Figure 3. Exemplified Capability Candidate Identification.

2) The business activity is IT supported (automatic) and is included by a semi IT-supported activity that is associated with at least one performer role or recursively included by an activity that is associated with at least one performer role.

Explanation: If an activity is semi IT supported only but performed by one or more performer roles, at least the parts that are IT supported should be provided by a service.

If it is expected that the capabilities perform autonomously mostly, i.e., without dependencies to other capabilities, the derivation of the service capability candidates is finished. If reusability is more important than autonomy, the following two rules are applied additionally:

3) The business activity is IT supported (automatic) and is included by more than one other business activity.

Explanation: This enables the sharing of this capability and thus increases the reusability.

4) The (composed) business activity is semi IT supported and is associated with more than one business performer role and includes at least two IT-supported or semi IT-supported business activities.

Explanation: The activity constitutes a composition of at least two IT-supported (automatic) activities. By selecting this activity as a capability, the automatic parts are provided as a reusable composition, even it is semi IT supported only. The provision of this composition increases the reusability. The subordinate activities are provided as service capability according to rule number two.

In a next step, the service candidates have to be determined. To encapsulate the identified capabilities into service candidates, our approach uses the distinction of entity services, task services and utility services as proposed in [1, 13]. To derive the service candidates and the encapsulated service capability candidates systematically, the following rules are applied. These rules only consider the prior as service capability candidate-identified business activities.

5) If a business activity is used by more than one business process and all of its included business activities are associated with exactly one and the same business entity or additionally with business entities that are included by the one using the composite aggregation, an entity service candidate for this (including) business entity is created. The service candidate is named after the particular business entity and the business activity is added as service capability candidate. The name of the capability equals the name of the business activity without the name of the business entity.

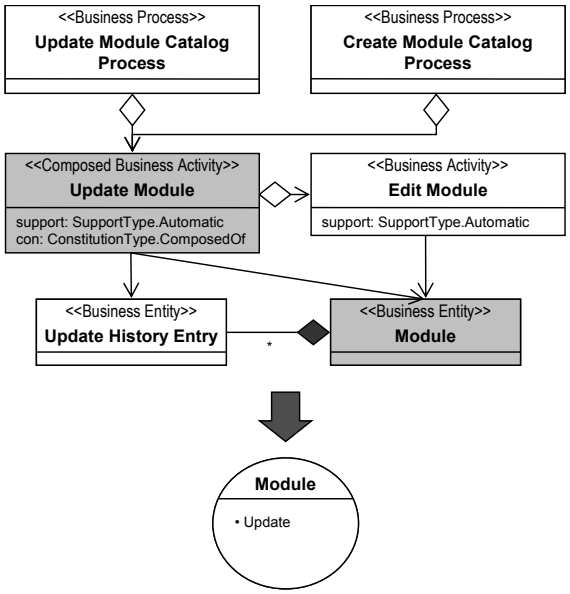

Figure 4. Exemplified Entity Service Candidate Identification.

Explanation: An entity service includes capabilities whose scope is exactly one business entity. Since business entities that are included using the composite aggregation cannot exist on their own, a separate service would not be autonomous, thus cannot be independently deployed, versioned, and managed [5]. For this reason, these entities are merged.

Figure 4 shows the exemplified identification of an entity service candidate. The prior as service capability candidate-identified business activity "Update Module" is included by two business processes. "Update Module" and the included business activity "Edit Module" both affect the business entities "Update History Entry" and "Module", and the business entity "Module" aggregates "Update History Entry" using a composite aggregation. Thus, an entity service capability candidate "Module" can be created including one service capability "Update". For the service candidate, the notation according to [1] is used.

6) If a business activity and all its included business activities are associated with at least two different business entities that are not included using the composite aggregation, a task service candidate is created. It is named after the business activity with the verb as prefix. One service capability candidate is added named after the verb of the activity.

Explanation: These capabilities should not be part of the corresponding entity service candidate, so as not to decrease the reusability of the entity service candidate. If the business entities are included using the composite aggregation, rule five applies.

Task services that do not represent business processes potentially can be merged if they are invoked by the same business process and affect the same business entities. The reason is that task services are less process-agnostic than entity services [1] and are bound to the scope of the parent business process. Thus, only task services that are invoked by the same business process and affect the same business entities should be merged, so as not to decrease the reusability. The name of the merged task service has to be individually revised.

7) If a business activity is associated with no business entity, a utility service candidate is created. All utility service candidate relevant business activities that are included by the same business activity constitute the same utility service candidate and are included as capability candidates. If a utility service candidate relevant business activity is included by more than one business activity, the parent business activity is considered until only one business activity or even business use case is reached. The utility service candidate is named after this, including business activity or business use case with the noun "Utilities" attached.

Explanation: Utility services do not influence business entities. To create reusable utility services, the capability candidates are grouped by their lowest common including business activities.

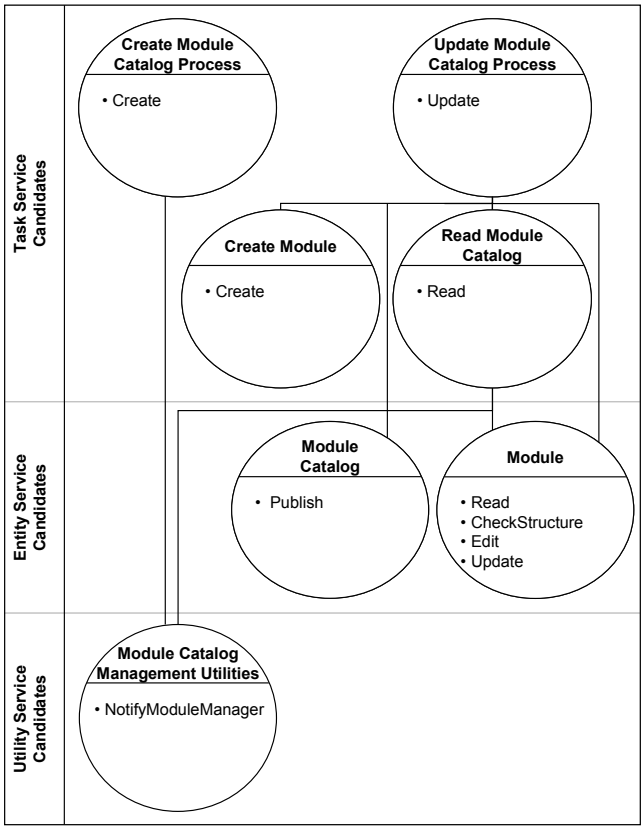

Figure 5. Service Candidates for the Module Catalog Management.

8) Each business process is transformed into one task service candidate representing the process. It is named after the business process and includes one service capability candidate named after the prefixed verb of the process.

Figure 5 illustrates the resulting set of service candidates and their hierarchy for the module catalog management scenario. This includes the prior identified entity service candidate "Module" and its service capability candidate "Update". The hierarchy of the service candidates is derived from the business model and the dependencies between the business activities. Since service capability candidates are derived from the business activities, their dependencies can be used to determine the hierarchy.

C. Formalization

Using a business model capturing the necessary business requirements for service derivation and the existence of a formalized meta model enable the formalization of the derivation rules. In the following, Object Constraint Language (OCL) [18] expressions are used to exemplarily describe rule 1 and rule 5. Therefore, some boolean attributes that describe whether a business activity represents a service capability candidate and whether a business entity represents an entity service candidate are assumed.

1) Formalization of rule 1:

```
context BusinessActivity
inv: support = SupportType::Automatic and
  performedBy->notEmpty()
  implies capabilityCandidate = true
```

2) Formalization of rule 5:

```
context BusinessActivity
def: allIncludes: Set(BusinessActivitiy) =
  self.includes->union(self.includes->
  collect(p | p.allIncludes())
def: agnostic: Boolean =
  BusinessProcess.allInstances()->exists(p1 |
  p1.activities->allIncludes()->includes(self) and
  BusinessProcess.allInstances()->exists(p2 | p1
  <> p2 and p2.activities->
  allIncludes()->includes(self)))

context BusinessEntity
def: allCompositeAggregates: Set(BusinessEntity) =
  self.entityAggregation[aggregates]->
  select(type = AggregationType::Composite)->
  union(self.entityAggregation[aggregates]->
  select(type = AggregationType::Composite)->
  collect(ea | ea.allCompositeAggregates()))
inv: affectedBy->exists(ba | ba.agnostic() and
  self.allCompositeAggregates()->
  includesAll(ba.allIncludes()->collect(affects)))
  implies entityServiceCandidate = true
```

The other rules can be formalized in a similar way. The usage of OCL allows the integration with existing modeling tools to support an automatic application of the derivation rules.

D. Embedding into an Entire SOA Development Process

The target of service-orientation is to support business processes of an organization. The rule-based service modeling focuses the creation of the service inventory blueprint and does not describe how it helps to realize business processes. Thus, this section shows the integration of the rule-based service modeling applied on the introduced business model into an entire SOA development process. This illustrates how a prior created service inventory blueprint can be combined with other existing approaches to improve the realization of business processes.

After the creation of the service inventory blueprint, the following steps are repeated for each business process: First, the business process is analyzed and modeled resulting in a business process model. In a next step, the business process is decomposed. With the existence of a service inventory blueprint the decomposition of a business process can be controlled. The business process has to be decomposed until each automatable activity matches a service capability candidate provided by a service candidate within the service inventory blueprint. This may require a transformation of terms to find an appropriate capability candidate. This step results in a so-called workflow model that only consists of activities that can be supported through services and their capabilities. The existence of the service inventory blueprint guarantees that all workflow models have a unified taxonomy and level of abstraction. In a last step, the necessary services are developed, if they haven't been already.

IV. Conclusion and Outlook

In this paper, we presented a rule-based service modeling approach applied to a customized business model to systematically derive the service candidates and their capabilities necessary to support the business. The business model captures relevant business requirements within one model and thus enables the unification of taxonomies and the level of abstraction. Additionally, it helps to identify redundant activities that would result in several services with the same capabilities during the service modeling process. The rules enable the derivation of a common ground for all business requirements. Combined with the prior defined meta model, the formalized rules using OCL expressions allow the integration with modeling tools. This enables the automatic service derivation from a prior created business model.

Our approach helps software architects to systematically identify the required services during the service modeling process. The embedding into an entire SOA development shows how our approach can be combined with other existing approaches to realize business processes. With the existence of a prior derived service inventory blueprint, the teams realizing single business processes have a guideline for how to decompose business processes. This reduces the risk to develop new services with new names and similar capabilities. Thus, the inventory blueprint helps to support the governance of a service-orientation. Additionally, since the business model focuses on the activities within an organization, the business processes with the concrete flows are not required to apply the rules. Thus, this approach can also be applied in scenarios in which the business processes are unknown, such as if software vendors want to provide services, even if they do not know the differing concrete processes of the customers.

We illustrated the application of our approach for the module catalog management scenario at the KIT. Instead of presuming services, the services were systematically derived and constitute a common ground for the entire module catalog management.

Our next steps are further work on how the resulting service inventory blueprint can be measured and evaluated regarding its design quality using software design metrics. Additionally, we work on further rules considering prior definable design goals. The approach itself is planned to be applied in further projects at the KIT.

References

[1] T. Erl, "SOA – Principles of Service Design", Prentice Hall, 2007. ISBN 978-0-13-234482-1.

[2] S. Cohen, "Ontology and Taxonomy of Services in a Service-Oriented Architecture", Microsoft Architecture Journal, 2007.

[3] P. Jamshidi, M. Sharifi, S. Mansour, "To Establish Enterprise Service Model from Enterprise Business Model", 2008.

[4] S. Jones, "Enterprise SOA Adoption Strategies", InfoQ Enterprise Software Development Series, 2006.

[5] J. Evdemon, "Principles of Service Design – Service Patterns and Anti-Patterns", MSDN Library, 2005.

[6] IBM, "RUP for Service-Oriented Modeling and Architecture, IBM Developer Works, http://www.ibm.com/developerworks/rational/downloads/06/rmc_soma/, 2006.

[7] S. Johnston, "Rational UML Profile for business modeling", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/5167.html, 2004.

[8] J. Heumann, "Introduction to business modeling using the Unified Modeling Language (UML)", IBM Developer Works, http://www.ibm.com/developerworks/rational/library/360.html, 2003.

[9] A. Arsanjani. "Service-oriented modeling and architecture - How to identify, specify, and realize services for your SOA", IBM Developer Works, http://www.ibm.com/developerworks/library/ws-soa-design1/, 2004.

[10] P. Kroll, P. Kruchten, "The Rational Unified Process Made Easy", a Practitioner's Guide to the RUP, Addison-Wesley, 2003.

[11] N. Fareghzadeh, "Service Identification Approach to SOA Development", Proceedings of World Academy of Science, Engineering and Technology, Volume 35, 2008.

[12] T. Erl, "SOA – Design Patterns", Prentice Hall, 2008. ISBN 978-0-13-613516-6.

[13] T. Erl, "Web Service Contract Design & Versioning for SOA", Prentice Hall, 2008. ISBN 978-0-13-613517-3.

[14] D. Krafzig, K. Banke, D. Slama, "Enterprise SOA: Service Oriented Architecture Best Practices", Prentice Hall International, 2005. ISBN 978-0-13-146575-6.

[15] M. Perepletchikov, C. Ryan, K. Frampton, H. Schmidt: "Formalising Service-Oriented Design", Journal of Software, Volume 3, February 2008.

[16] Object Management Group, "Unified Modeling Language, Superstructure", Version 2.2, 2009.

[17] Object Management Group, "Business Process Definition MetaModel", Version 1.0, Volume 2, Process Definitions, 2008.

[18] Object Management Group, "Object Constraint Language", Version 2.0, 2006.