Designing Reusable Management Services

Ingo Pansa[1], Christoph Leist[2], Matthias Reichle[2], Sebastian Abeck[1]
Cooperation & Management
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
[1]{pansa, abeck}@kit.edu
[2]{christoph.leist, matthias.reichle}@student.kit.edu

Abstract— **Reusing functionality is one preferable requirement in today's engineering of distributed systems. Focusing IT Management systems as a key enabler to modern service-oriented systems, reusing management functionality can be achieved by applying the principles of service-orientation to support the construction of reusable management services. Thus, in order to construct these management services aligned with certain design quality, estimating the possible degree of reusability during analysis and design steps is required in order to support certain design decisions. Existing approaches targeting the design of management services do not take reusability into account explicitly, wherefore the proposed solutions seem to be hard to adopt if requirements to that system change. In this paper, an overall approach based on domain modeling is presented, supporting the design of management services by explicitly defined reusability metrics. The approach is exemplified by designing management services for a typical Incident Management scenario in which we outline the value of domain modeling for creating reusable design blueprints.**

Keywords - management service design; reusability; domain model

I. INTRODUCTION

As nowadays software systems grow in complexity, decoupling different parts of the systems is one of the most desired characteristics that system engineers follow. Different approaches have been proposed, starting with the early Client/Server-Architectures followed by CORBA [9] in the 90's up to Web Service-based Architectures in the beginning of the new century. While all these approaches have major differences in how to structure the proposed architectures, they have some basic principles in common, of which the reusability of existing software artifacts seems to be one of the most important.

Focusing Service-oriented Architecture (SOA) [15], reusability of software artifacts is reflected in the existence of clearly defined service interfaces that hide details of the service implementation [27]. These service interfaces are expected to align with business process requirements thus supporting a basic reusability on a coarse granular level. Furthermore, standardized technologies such as Web Services or Universal Description, Discovery and Integration (UDDI) [33] are utilized to realize technical aspects of reusability.

While there seems to exist a common agreement of how to describe the concept of and specify formal metrics for reusability at least in Component-based Software Engineering (CbSE) [23, 25], a clear understanding of reusability in Service-oriented Software Engineering (SoSE) has not yet been reached. Although initial work exists that regards reusability as a key concept in SOSE [23, 26, 27], the definition of formal metrics that can directly be used within typical modeling languages supporting service-oriented analysis or service-oriented design (e.g., SoaML [17]) is still missing. To impair this situation, reusability becomes important considering the different viewpoints towards SOA.

To address this situation, this paper delivers initial contributions: First, we introduce refined aspects of an abstracted development process for SOSE in which we identify activities that deal with reusability and discuss characteristics of SoaML elements relating to reusability. Second, we present selected metrics measuring reusability of specific service analysis or service design models. The presented metrics are based on common agreement of how to describe and measure reusability of software artifacts on a conceptual level [4, 18, 19, 20]. Third, we demonstrate the application of these metrics in a real world scenario dealing with the construction of reusable services for a distributed management system that is based on reusable management services [2, 3, 6, 7].

The remaining parts of this paper are structured as followed: in Section 2, we outline the background of reusability in service-oriented architecture and summarize related work. Section 3 presents a typical service-oriented development process that is focused to consider aspects of designing reusable management services. In Section 4, the main contribution of this paper is introduced: we discuss three different aspects of reusability of management services in detail and present formal metrics to evaluate the respective aspect. Section 5 embeds the presented metrics in a real world development process considering management services supporting a typical Incident Management process. We chose to demonstrate the applicability of our approach within a very special scenario as future management systems will greatly benefit from applying service-orientation [10, 11, 12, 13, 14, 22, 24] thus require proper designed management services according to reusability aspects.

 ISBN: 978-1-61208-152-6

Finally, Section 6 summarizes the results of this paper and presents some ongoing work that can complement the proposed approach.

II. BACKGROUND AND RELATED WORK

As a special instance of distributed information systems, constructing software systems supporting IT Service Management (ITSM) follows similar principles. Considering the construction of such systems, the main challenges that distributed management is faced with are named in [10, 11, 12, 22, 24]. Although there exist a few holistic approaches considering technical aspects of distributed management based on web services [30, 31], only a few papers have been published dealing with more process-oriented aspects of integrated management systems [6, 7, 13, 14, 32]. One can conclude that, although initial work towards standardized and reusable management services has been performed, a revision of these approaches contributing to the process of Service-oriented Software Engineering (SoSE) is necessary. As we currently observe a shift towards web-based usage of dynamic IT Services ("Cloud Computing") with the broader adoption of flexible service infrastructures by the business, this holds even more. Standards such as ISO/IEC20000-1:2005 [5] only serve as a starting point.

According to [4], reusing software is the process of creating software systems from existing software rather than building software systems from scratch. Thus, from the perspective of a software designer tasked to create a collaborative system, using for instance deployed artifacts is a building block to create a system that fulfills requirements that are subject-specific. Focusing this generally applicable assumption to the challenge of creating a collaborative system supporting management activities, applying the principles of service-orientation perfectly seems to fit these requirements.

Initial work has been published lately considering design issues of reusable services [23, 26], however, investigating reusability of software artifacts is a much more older research topic and is based on concepts that were introduced at the NATO Software Engineering Conference in 1968 [35]. Many research efforts have been undertaken to address different aspects of reusability (e.g., in Component-based Software Engineering [36]), including extensive survey papers [4] that conclude the then leading insights.

Following Krueger, four different criteria have to be regarded considering reusability: *abstraction*, *selection*, *specialization* and *integration* [4]. While these concerns are of very generic nature, Erl introduces four extra criteria focused on designing service-oriented software artifacts [27]: *agnostic from business processes*, *generic business logic*, *generic service contract* and *concurrency*. However, the presented criteria in [27] are discussed on a conceptual level without any formal defined foundation. Besides many more, Poulin addresses reusability focused on object-oriented software design [19] by focusing the criteria *cohesion*, *autonomy*, *usefulness* and *complexity*. While the discussed criteria serve as a direct foundation to investigate criteria for service-oriented design, Poulin mainly focuses on business-related aspects thus considering economic measures rather than engineering measures.

Apart from the beforehand named criteria, generic aspects desired when constructing software components such as *complete* operation *sets* or *disjoint operation sets* are mainly motivated by practical concerns derived from experience we observed in several development projects. While the criterion *complete operation sets* aims at reducing future development efforts by explicitly extending service design based on predefined patterns, focusing *disjoint operation sets* tries to prevent the definition of redundant operations thus leading to side effects when changing existing service logic.

III. A SERVICE-ORIENTED DEVELOPMENT PROCESS

Designing distributed software systems is a highly complex issue that involves several different stakeholders. Focusing Service-oriented Architecture (SOA), some generic steps and models can be identified that are independent of concrete development process models. In order to utilize the metrics framework presented afterwards, in this chapter we briefly discuss an abstracted view of such typical development models. The abstracted view is presented in means of a scenario within a typical IT Service Provider (ITSP) that aims at automating its management processes based on its existing management tools. The integrative artifacts typically are implements using web services. Figure 1 shows for an overview of the assumed scenario.

As the proposed standard language for modeling service-oriented software systems already is adopted by tool vendors, the entire development process is supported by SoaML [17] for modeling services and OWL [37] for the definition of ontologies [8]. We propose to utilize OWL ontologies for defining domain models as according to [1] this approach brings several advantages focusing model-driven software development.

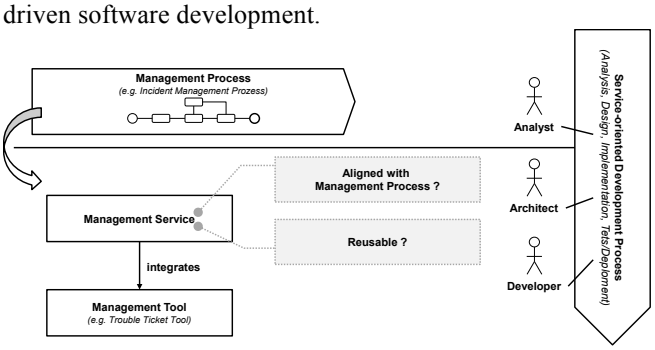

Figure 1. Service-oriented integration of managemen tools

The service-oriented design process discussed here is derived from and aligned with established software development processes and thus consists of the four phases

service-oriented analysis, *service-oriented design*, *service implementation* and *deployment*.

The goal of service-oriented analysis is to capture the characteristics and requirements of the problem domain and transform them into a set of service candidates providing the necessary functionality.

To accomplish this, the analyst will first specify a domain ontology, serving not only as the basis for the following analysis steps but also as a reference point for activities throughout the entire development process, such as the evaluation of reusability conducted mainly in the design phase. The domain ontology is an extension of a common and binding domain meta model [2, 3], ensuring consistent syntax and semantics across multiple projects and development teams.

The next step is the specification of the high-level system behavior through the definition of formal business process models that refer to the concepts found in the domain model. Focused in our domain, we define management processes as special instances from generic business processes. From these management process models, service candidates will be derived according to the rules defined in [2], that, represented as SoaML Capabilities, mark the transition to the service-oriented design phase.

The rule-based transition from service candidates to abstract (e.g. platform-independent) service interfaces constitutes the first step in the development of the service interface model. It is followed by the specification of service contracts, participants and the overall architecture.

Our evaluation of the resulting services' reusability mainly takes place towards the end of the analysis phase and early in the design phase and is based on the service candidate and service interface models. This way, achieving high levels of reusability should become more likely, since the effort required to modify analysis or design models is relatively small compared with the modification of fully implemented software.

In the implementation phase, the abstract service interfaces are concretized using platform-specific interface definition languages such as the Web Services Description Language (WSDL) [28]. Basic services are realized through implementation in code or through integration of pre-existing tools; for composite services mechanisms like the Business Process Execution Language (BPEL) [29] may be used.

These activities however, as well as the subsequent deployment phase, are not covered by our research.

IV. REUSABILITY OF MANAGEMENT SERVICES

This section explores three of the aforementioned criteria for the reusability of services in greater detail and tries to establish a formal basis for their evaluation in concrete scenarios. The presented criteria are based on previously published work that, although targeting Object-oriented or Component-based Software Engineering, refines these approaches by explicitly addressing characteristics of a service-oriented design process.

A. Classification

To be able to discover the services to be reused in a specific context is essential for the process of selection and thus for reusability itself. In other words: "To reuse a software artifact effectively, you must be able to 'find it' faster than you could 'build it'." [4].

Classification is the non-technical aspect of discoverability; the technical aspect being the existence of some kind of service repository (such as Universal Description, Discovery and Integration (UDDI) [33]) supporting the actual retrieval of services.

While one can locate the services needed in a given scenario based on their name (which is, in fact, greatly facilitated by adhering to naming conventions), an extensive classification allows for a more precise search. The proposed classification categorizes the developed services according to the structure of the underlying domain model and thus allows us to locate and compare services based on a variety of different characteristics. Classifications can be defined in SoaML models using so-called Categories, that are mainly an extension of the OMG-defined Reusable Asset Specification (RAS) [16]. A SoaML Category contains several different SoaML Categorization elements that can be used to define a certain aspect.

Although we evaluate the reusability of services during design time, in order to be able to locate the implemented services the classification must pertain to them as well. We therefore assume that the SoaML Categorization elements used to classify design related artifacts are transformed into an appropriate semantic annotation of the resulting concrete service interfaces, such as Semantic Annotations for WSDL (SAWSDL) [34].

The classification dimensions for service candidates (represented by SoaML Capabilities) and service interfaces (represented by SoaML ServiceInterfaces) are shown in Table 1.

TABLE I. CLASSIFICATION DIMENSIONS

Symbol	Description
MST	*Management Service Type* Type of the service (Basic or Composed)
MCT	*Management Capability Type* Type of the specified capability (Provided or Required; only applies to service candidates)
MAT	*Management Area Type* Management Area the service belongs to
ME	*Management Entity* Entity the service operates on

To gain a measure for the extent of classification for a given service candidate MSC_x we divide the amount of classification dimensions associated with the service ($\mathrm{AoCD}(MSC_x)$) by the total number of applicable classification dimensions ($\mathrm{ToCD}_{\mathrm{SC}}$). Equation (1) defines the

ratio of classification dimensions, and (2), (3), (4) and (5) define several helper functions.

$$\mathrm{RoCD}_{\mathrm{SC}}(MSC_x) = \frac{\mathrm{AoCD}(MSC_x)}{\mathrm{ToCD}_{\mathrm{SC}}} \quad (1)$$

where

$$\mathrm{AoCD}(MSC_x) = \sum_{\forall i} \mathrm{Ex}(CD_i, MSC_x) \quad (2)$$

with

$$\mathrm{Ex}(CD_i, MSC_x) = \begin{cases} 1, \text{if } \exists c \in CD_i \text{ associated with } MSC_x \\ 0, \text{otherwise} \end{cases} \quad (3)$$

and

$$\mathrm{ToCD}_{\mathrm{SC}} = |\{CD_i \mid \forall i\}| \quad (4)$$

$$\mathrm{ToCD}_{\mathrm{SI}} = |\{MCT, MST, MAT, ME\}| = 4 \quad (5)$$

The extent of classification for the actual service interface is calculated accordingly. Equation (6) gives the definition of the ration of classification dimensions for an actual service interface and (7) defines the needed dimension set.

$$\mathrm{RoCD}_{\mathrm{SI}}(MSI_x) = \frac{\mathrm{AoCD}(MSI_x)}{\mathrm{ToCD}_{\mathrm{SI}}} \quad (6)$$

with

$$\mathrm{ToCD}_{\mathrm{SI}} = |\{MST, MAT, ME\}| = 3 \quad (7)$$

An RoCD value of 1 implies an optimal classification coverage for the evaluated artifact (service candidate or service interface) in that there exists an association with at least one classification element from each available classification dimension. Values closer to 0 on the other hand indicate a relatively poor classification coverage, which is not desirable.

B. Complete operation sets

Many commonly used interface operations appear in groups, such as the well-established CRUD pattern or operation pairs like *open/close*. Completeness regarding such patterns benefits reusability, because it is very likely that, once one of the operations contained in a group is needed, all of them will be at some point.

Ignoring completeness patterns can lead to uncontrolled extension of service interfaces resulting in a loss of cohesion (if additional functionality is assigned to a separate interface) or even non-disjoint functional contexts (if functionality is unintentionally duplicated).

The sub-criteria for the completeness of data-centric services — in our context called *Entity Services* [27] — are the existence of a *Create*, *Read* and *Update* method, captured by (8):

$$\mathrm{Ex}(CO, MSC_x), \mathrm{Ex}(RO, MSC_x), \mathrm{Ex}(UO, MSC_x) \quad (8)$$

Since ISO/IEC20000-1:2005 [5] — on which the motivating example is based — does not permit the deletion of records, we do not consider the existence of a *Delete* operation necessary for interface completeness.

Thus, the measure for the completeness of an Entity Service is defined in the following equations (9) and (10)

$$\mathrm{RoC}_{\mathrm{SC}}(MSC_x) = \frac{\mathrm{Ex}(CO,MSC_x)+\mathrm{Ex}(RO,MSC_x)+\mathrm{Ex}(UO,MSC_x)}{3} \quad (9)$$

$$\mathrm{RoC}_{\mathrm{SI}}(MSI_x) = \frac{\mathrm{Ex}(CO,MSI_x)+\mathrm{Ex}(RO,MSI_x)+\mathrm{Ex}(UO,MSI_x)}{3} \quad (10)$$

where a value of 1 indicates completeness of the service regarding the CRU pattern and values below 1 indicate lacking completeness.

It should be mentioned that the concept of completeness can also be applied to data types (e.g., by demanding the existence of an ID attribute), although this paper does not further investigate this.

C. Disjoint operation sets

Like other, more "traditional" software systems, service-oriented architectures depend on the separation of concerns and on clearly defined functional borders. Those concepts can have a positive effect on reusability insofar as they structure the collection of available services and help to alleviate the problems arising from duplicated functionality such as productivity losses and potential incompatibilities and access conflicts.

In this context, one policy that is both easy to define and easy to enforce is the exclusive data access of Entity Services, implying that the access to one class of entities is to be provided by the corresponding Entity Service alone.

A statement about two services being disjoint can be made by determining the overlap of their respective operations. Assuming we have a means to decide whether two operations are functionally equivalent, defined by (11)

$$\mathrm{Cov}(O_1, O_2) = \begin{cases} 1, \text{if } O_1 \text{ and } O_2 \text{ are equivalent} \\ 0, \text{otherwise} \end{cases} \quad (11)$$

we can derive a measure for disjoint operation sets of services as followed in (12):

$$\mathrm{AoSSO}_{\mathrm{SC}}(MSC_x) = \sum_{i=0}^{i<|\{O_x\}|} \sum_{j \neq x} \sum_{k=0}^{k<|\{O_j\}|} \mathrm{Cov}(O_{x,i}, O_{j,k}) \quad (12)$$

As this returns the total number of one service's operations also found in other services, a value of 0 (indicating completely disjoint services) should be targeted.

V. DESIGNING REUSABLE SERVICES FOR INCIDENT MANAGEMENT

The following example is an excerpt from a project involving the introduction of an IT-supported incident management process. It shows the refinement of one basic service through the analysis, design, and implementation phases of the development process discussed in Section 3.

Figure 2 shows the relevant parts of a domain ontology created in accordance with ISO/IEC20000-1:2005 [5]. As depicted, two management activities (*RecordIncident* and *CreateIncidentRecord*) are regarded. The two management activities belong to different types of management activities but can be ranged in the management area IncidentManagement. The basic management activity *CreateIncidentRecord* has access to the management entity *IncidentRecord*. The presented domain ontology was defined using OWL.

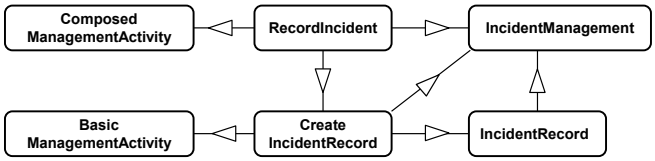

Figure 2. Excerpt from domain ontology

For the sake of simplicity, we exclusively consider the management activity for recording an incident, requiring the basic capability to create an *IncidentRecord* entity. The service operation *CreateIncidentRecord* is assigned to a SoaML Capability named *IncidentRecordService* and categorized as *RequiredManagementCapability* (denoting a needed as opposed to an already existing service) (see Figure 3).

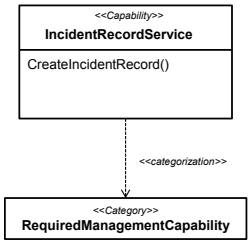

Figure 3. Preliminary service candidate

A preliminary evaluation of the service candidate's reusability shows that the criteria complete operation sets and disjoint operation sets are not fulfilled yet, as indicated by the following applications of the presented metrics:

$$\mathrm{RoCD}_{\mathrm{SC}}(IRS) = \frac{0+1+0+0}{4} = \frac{1}{4}$$

and

$$\mathrm{RoC}_{\mathrm{SC}}(IRS) = \frac{1+0+0}{3} = \frac{1}{3}$$

Consequently, the SoaML Capability is further categorized as a *ManagementBasicService* (it is, in fact, a *ManagementEntityService*, a special kind of *ManagementBasicService*), operating on *IncidentRecord* entities and belonging to the area of *IncidentManagement*. Furthermore, it is completed with respect to the CRU pattern by adding the operations *ReadIncidentRecord* and *UpdateIncidentRecord*, resulting in

$$\mathrm{RoCD}_{\mathrm{SC}}(IRS) = 1$$

and

$$\mathrm{RoC}_{\mathrm{SC}}(IRS) = 1$$

The modified service candidate can be seen in in Figure 4.

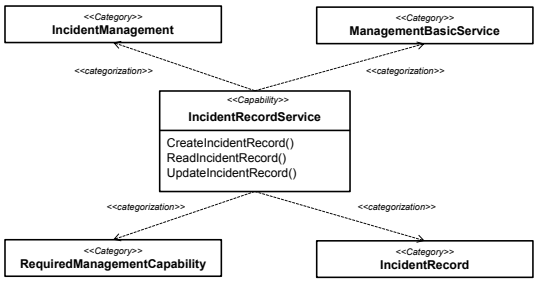

Figure 4. Modified service candidate

Since this example focuses on one single service, its operations cannot be compared to those of other services. Following the data sovereignty policy on the other hand ensures that operations managing the lifecycle of *IncidentRecords* are only found on *IncidentRecordService*. It follows that

$$\mathrm{AoSSO}_{\mathrm{SC}}(IRS) = 0$$

The ServiceInterface named *IncidentRecordService* (Figure 5) is derived from the refined Capability, whose classification it shares (with the exception of *RequiredManagementCapability*, which only applies to service candidates). Its operations are provided with appropriate parameters and return values.

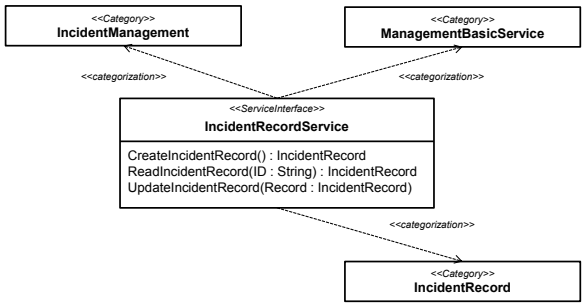

Figure 5. Service Interface

Not surprisingly, the results of the reusability evaluation are the same as for the service candidate:

$$\mathrm{RoCD}_{\mathrm{SI}}(IRS) = 1$$

$$\mathrm{RoC}_{\mathrm{SI}}(IRS) = 1$$

$$\mathrm{AoSSO}_{\mathrm{SI}}(IRS) = 0$$

The actual implementation of the *IncidentRecordService* will be achieved by the adapter-based integration of *Mantis BugTracker* [21], a trouble ticket tool currently in use as a standalone solution.

VI. CONCLUSION AND OUTLOOK

Reusing existing software assets seems to many researchers a kind of Holy Grail when engineering complex and distributed information systems. While a couple of different approaches and paradigms have been proposed in the past (ranging from simple Client/Server computing to up to Component-based Software Engineering (CbSE)) to address general problems when reusing software assets, many issues still remain. Grounded on the simple statement, that reusing does not come for free [19], special attention has to be paid not only within the design process but also when selecting appropriate models and modeling techniques.

As nowadays information systems have to be aligned with business processes, the requirements for reengineering business logic can directly be derived from the business process perspective. Considering Service-oriented Architectures (SOA) to realize these process-oriented information systems, the systems elements that implement SOA have to be aligned with the business processes. Thus, reusability of services has to be regarded from the perspective of the technical-independent processes. Existing approaches do not consider process requirements explicitly when targeting the design of reusable services but mainly focus on technical details.

To address this issue, in this paper we deliver several contributions. First, we refine a generic development process for service-oriented analysis and design and outline development tasks that are supported by different models and modeling techniques that focus the reusability of the to-be-designed artifacts. The presented development approach extends and refines work that was previously published by our research group [2, 3]. Second, we discuss an assorted selection of different aspects of reusability considering service orientation and present three different aspects that are formalized using a conceptual metrics framework. The presented metrics can be applied to any kind of service analysis or design models if they are defined using SoaML. Furthermore, we outline the advantage of using an OWL-based domain ontology for directly influencing the quality of service design. Using domain ontology has several advantages [2, 3]. As we expect that reusability of services can only be discussed within clearly defined domains, we present an application example of our approach within the context of designing web service-based services for a process-oriented management system supporting IT Service Management Processes. A third contribution therefore devotes to a typical Incident Management process and demonstrates the application of both the presented development process and the introduced metrics framework, resulting in a set of management services that are designed along special design characteristics.

Although service-oriented computing inherently is predestinated for building software systems based on existing assets, it seems remarkable that existing approaches mainly focused on technical details. Considering the contributions we deliver in this paper, we address a more conceptual perspective, but further work has to be performed. As we mainly focused on generic issues targeting design related aspects of services reusability, a more formal approach that is independent of certain domains could greatly enhance software engineering. Utilizing model-driven techniques could not only decrease engineering round trip times, but also increase the quality of resulting systems implementation.

REFERENCES

[1] D. Gasevic, D. Djuric, and V. Devedzic, Model Driven Engineering and Ontology Development, Springer, Heidelberg, 2006.

[2] I. Pansa, F. Palmen, S. Abeck, K. Scheibenberber, "A Domain-driven Approach for Designing Management Services", SERVICECOMPUTATION2010, Lisbon, 2010.

[3] I. Pansa, P. Walter, K. Scheibenberger, and S. Abeck, "Model-based Integration of Tools Supporting Automatable ITSM Processes", IEEE/IFIP Network Operations and Management Symposium Workshops (NOMS Wksps), 2010, Page(s): 99 – 102.

[4] C.W. Krueger, "Software reuse", *ACM Computing Surveys (CSUR)*, vol. 24, no. 2, pp. 131-183, 1992.

[5] ISO/IEC20000-1:2005, Information technology – Service management – Part1: Specification, International Standards Organization (ISO), 2005.

[6] G. Aschemann and P. Hasselmeyer, „A Loosely Coupled Federation of Distributed Management Services", Journal of Network and Systems Management, Vol 9, No. 1, 2001.

[7] N. Anerousis, „An Architecture for Building Scalable, Web-Based Management Services", Journal of Network and System Management, Vol. 7, No 1., 1999.

[8] T. Gruber, „A translation approach to portable ontology specifications", Knowledge Acquisition, Vol. 5, Issue 2, pp. 199-220, 1993.

[9] M. Henning, „The Rise and Fall of CORBA", Communications of the ACM Vol. 51 No. 8, pp. 52-57, 2008.

[10] P. Kumar, „Web Services and IT Management", ACM Queue, Volume 3 Issue 6, 2005.

[11] V. Machiraju, C. Bartolini and F. Casati, „Technologies for Business-Driven IT Management", in Extending Web Services Technologies: the Use of Multi- Agent Approaches, Kluwer Academic, 2004.

[12] A. Moura, J. Sauve and C. Bartolini, „Research Challenges of Business-Driven IT Management", 2nd IEEE/IFIP International Workshop on Business- Driven IT Management, pp.19-28, 2007.

[13] C. Mayerl, F. Tröscher, and S. Abeck, „Process-oriented Integration of Applications for a Service-oriented IT Management", The First

 ISBN: 978-1-61208-152-6

International Workshop on Business-Driven IT-Management, BDIM'06, pp. 29-36, 2006.

[14] C. Mayerl, T. Vogel and S. Abeck, „SOA-based Integration of IT Service Management Applications", 2005 IEEE International Conference on Web Services (ICWS 2005), 2005.

[15] Organization for the Advancement of Structured Information Standards (OASIS), Reference Model for Service Oriented Architecture, Version 1.0, OASIS, August 2006.

[16] Object Management Group (RAS), Reuasable Asset Specification, Version 2.2, OMG, November 2005.

[17] Object Management Group (OMG), Service-oriented architecture Modeling Language (SoaML) - Specification for the UML Profile and Metamodel for Services (UPMS), FTF Beta 1, OMG, April 2009.

[18] R. Prieto-Diaz and P. Freemann, „Classifying Software for Reusability", IEEE Software, Vol. 4, Issue 1, pp. 6-16, 1987.

[19] J. Poulin, „Measuring Software Reuse", Addison Wesley Publishing Group, 1997.

[20] Ruben Prieto-Diaz, „Status Report Software Reusability", IEEE Software, Vol. 10, Issue 3, pp. 61-66, 1993.

[21] Mantis Bug Tracker, 2011; http://www.mantisbt.org/ (last vistited: 19- 02-2011).

[22] J. Sauvé, A. Moura, M. Sampaio, J. Jornada and E. Radziuk, „An Introductory Overview and Survey of Business-Driven IT Management", 1st IEEE/IFIP International Workshop on Business-driven IT Management (BDIM 2006), 2006.

[23] A. Sillitti, and G Succi, „Reuse: From Components to Services", High Confidence Software Reuse in Large Systems (2008), pp. 266-269, 2008.

[24] V. Tosic, „The 5 C Challenges of Business-Driven IT Management and the 5 A Approaches to Addressing Them", The First IEEE/IFIP International Workshop on Business-Driven IT Management, 2006.

[25] H. Washizaki, H. Yamamoto, and Y. Fukazawa, „A Metrics Suite for Measuring Reusability of Software Components", Proceedings of the 9th International Symposium on Software Metrics, 2003.

[26] J. Wang, J. Yu, P. Falcarin, Y. Han, and M. Morisio, „An Approach to Domain- Specific Reuse in Service-Oriented Environments", Proceedings of the 10th international conference on Software Reuse: High Confidence Software Reuse in Large Systems, 2008.

[27] T. Erl: SOA, Principles of Service Design, Prentice Hall, 2008.

[28] World Wide Web Consortium (W3C), Web Service Description Language (WSDL) Version 2.0 Part1 Core Language, W3C Recommendation, 2007.

[29] Organization for the Advancement of Structured Information Standards (OASIS), Web Services Business Process Execution Language (WS- BPEL), Version 2.0, OASIS, April 2007.

[30] Organization for the Advancement of Structured Information Standards (OASIS), Web Services Distributed Management: MUWS Primer, OASIS, Februar 2006.

[31] Distributed Management Task Force (DMTF), Web Services for Management (WS-Management) Specification, Version 1.1.0, DMTF, 2010.

[32] G. Tamm and R. Zarnekow, „Umsetzung eines ITIL-konformen IT-Service-Support auf der Grundlage von Web-Services", Wirtschaftsinformtik 2005, pp. 647-666, 2005.

[33] Organization for the Advancement of Structured Information Standards (OASIS): Universal Description, Discovery and Integration (UDDI), Version 3.0.2, Oktober 2004.

[34] World Wide Web Consortium (W3C): Semantic Annotations for WSDL and XML Schema, W3C Recommendation, 2007.

[35] P. Naur and B. Randall, "Software Engineering; Report on a conference sponsored by the NATO Science Committee", Garmisch, 1968.

[36] B. Meyer, "Reusability: The case for Object-oriented design", in Frontier Series: Software Reusability, Vol. 11 – Applications and Experience, 1989, pp. 1-33.

[37] World Wide Web Consortium (W3C), Web Ontology Language (OWL), W3C Recommendation, 2004.

 ISBN: 978-1-61208-152-6