Improving the Extent and Reach of Service Oriented Systems

Martin Henkel

Licentiate Thesis at Stockholm University, Sweden 2006

Improving the Extent and Reach of Service Oriented Systems

Licentiate Thesis

Martin Henkel

Department of Computer and System Sciences
Stockholm University and Royal Institute of Technology

August 2006

Abstract

Service oriented systems are built as a set of loosely coupled software services – each service representing a run-time software unit provisioned by an independent party. A vision driving the development of service oriented systems is to interconnect existing and future software resources, both within an organisation and across organisations. The success of service oriented systems relies on the management and combination of a large *extent* of services. Furthermore, it is of importance to improve the *reach* of services such that software services can be interconnected across organisational boundaries - thereby supporting virtual enterprises. This thesis contributes with instruments that improve both the reach and extent of systems based on software services. To improve the extent and reach of service use, two separate perspectives are presented - business and technical. These two perspectives are joined by the specification of a set of process transformation patterns and a set of transformation rules that need to be adhered to in order to keep the perspectives aligned. The extent of service use is further improved by analysing a set of architectural styles that can aid the alignment of the technical and business perspectives. Furthermore, the use of aspects is examined as a way to externalize previously internal software services, thereby improving the reach of service oriented systems. The presented instruments are structured in a framework describing their applicability in a set of categories of service use.

Table of Contents

Part I

Part II – Included Papers

1 Introduction

Service oriented computing has been touted as a new computing paradigm. In the centre of this paradigm is the view that future software systems will be built with services as the fundamental building block. The reason for the rising interest in the concept of service oriented systems is multi-faceted. One reason is based on the view that service oriented computing provides a modular way of building new systems. This makes service oriented computing a natural step of evolution from object-oriented and component based computing. Another aspect that makes services interesting is that they provide the basis for a run-time integration of information systems, both within an organization and across organizational boundaries. From an integration perspective, services are the interface to system resources residing in internal or external systems. Structuring systems as services and enabling the interconnection of them thus makes it possible to support networked or "virtual" enterprises [1]. The use of services both for cross-enterprise integration and as a building block for internal systems is enabled by the raised abstraction level that services provide [2] – commonly a software service is designed to support business functions or part of a business process [3].

Even though software services are modelled in accordance with business functions or business processes they are still exposed to technical considerations when deployed and managed in a technical environment consisting of legacy systems, communication protocols and message formats. Successfully building service oriented systems is thus not only about mapping business requirements to software modules. What is required is that services are examined and understood both from a *technical perspective* and from a *business perspective*. Understanding the business and technical perspectives of services enable us to construct new abstractions on the business level, while enabling technical integration in technical environments.

As the number of software services that an organisation requires increases, the importance of coordinating these services both on the business and technical levels becomes apparent. For example, having a large amount of services, all exchanging messages, make it necessary to define structured rules for message routing, translation and sequencing. These kinds of coordination issues can be handled by using executable process description languages, such as Business Process Execution Language for Web Services, BPEL4WS [4]. Composing existing services into new services by using process flows is a key to handling complex service dependencies both within and across organizations.

This thesis provides contributions on how to improve both cross-enterprise service use (reach) and the large scale use of service (extent of service use). The contribution, presented in four self-contained papers (Paper A-D), is built on three distinct aspects of software services:

- The concept of service as a software module
- The combination of business and technical perspectives on services
- The combination of processes and services

These aspects work as a foundation for the presented work. As a background to the presented work Section 1.1 contains a definition of software services, while the business

and technical perspectives of services are introduced in Section 1.2. The relationship between services and processes is briefly introduced in Section 1.3.

1.1 Services as Software Modules

From both a software perspective and from a business perspective, a service can be defined as *well defined work that can be offered by a provider to a consumer*. In a business scenario the provider and consumer are usually economically independent actors, while *for software services the provider is a software system and the consumer is either another software system or an (human) actor*. Web services based on the web service technologies WSDL [5] and SOAP [6] are an example of software services that are designed to be consumed by other systems. A software service is simply a provided module that represents a well defined functionality.

The concept of software services as modules differs from previous module concepts such as objects and components in both granularity and use. The basic concept of using modules to build systems is far from new. Modules are commonly used to improve the management, comprehension, flexibility and deployment of a system [7]. The concept of modularity is a strong driver behind service oriented computing [1]. However, the concept of services differs with respect to granularity, abstraction, deployment and run-time properties from other concepts of software modules, such as those found in object-orientation or component based computing. Firstly, a service is commonly considered to encompass more functionally compared to components and objects. Secondly, instead of being designed to match single business concepts (object orientation), or groups of business concepts (components, [8]), services are put on a higher level of abstraction [2], commonly modelled at the level of business functions or business processes.

Compared to modular concepts, such as objects and components, software services is commonly associated and used with a clear separation of consumer and provider responsibilities. What distinguishes services is that they are considered to be provided at run-time by a party [9]. This means that the service provider not only is responsible for the service development but also for the deployment and the run-time environment of the service.

The raised granularity and the clear separation of consumer and provider perspectives makes services particularly useful for software supported business integration.

1.2 Services from Software and Business Perspectives

When designing software services, the alignment between business and technical requirements is a key concern. Services only defined from a pure business perspective cannot commonly be executed "as-is" due to constraints in existing software systems.

From a business perspective, the design of services needs to consider business requirements i.e. the design should closely resemble the business operations, events and message exchanges that exist in the enterprise. This means that the services reflect the pure business perspective, as the focus is to "automate the business".

The design of software services must also consider a technical perspective, taking into account the future technical context of the service. In a majority of organizations, services have to be aligned with existing legacy systems and existing technologies. These systems

may impose specific requirements. For instance, it might be the case that parallel activities must be realized as sequential because of execution dependencies of the back-end systems. Another example is that there might not exist systems that support the activities of the software service in their current form. As result, to realize a service, the designer must consider the behaviour of existing systems as well as technology limitations. A final software service will thus reflect both the business and technology perspectives.

1.3 Services and Processes

The notion of *process* is relevant to describe both the interconnection of services, as well as describing the implementation of complex services. Executable processes are focused around describing dynamic aspects, such as the execution order and synchronization of activities. As defined in section 1.1, services are rather run-time modules provided by a service provider, and used by service consumers. Thus, while the concept of service is used to model the static aspects of a system, the concept of processes is used to describe the dynamic aspect.

Both from a technical and from a business perspective the combination of processes and services is interesting. From a business perspective, the use of processes to describe the interaction of services is beneficial because it allows the process designer to raise the abstraction level up to the business level. Ideally, process models describing the business can be used as a foundation for the design and execution of software services [10]. On a technical level process execution languages offer the ability to combine services with advanced flow synchronizations that would otherwise be difficult to implement. So, both from a business and a technology perspective, the combination of services and processes is important.

Furthermore, processes can be used to build composite services - services that are created by combining existing services. As the number of available services increases, there is increased benefit of and possibility to build new services by combining existing services. This way of building new services is called service composition [11]. Service compositions consist of calling other services and combining the result. Thus these compositions are best described as a sequence of interactions with other services. These sequences of interactions are possible to model as the process flows.

2 Research Problem

Using services in a large scale to interconnect internal as well as external resources puts new requirements on both the ability to coordinate large amount of services and the ability to coherently handle both internal and external services. The requirements, and thereby the problems to be solved, can broadly be categorised as *service extent* - the ability to handle large amounts of services, and *service reach* - the ability to connect to external services.

An increased *extent* of service use relies on that a large amount of services can be structured and interconnected. The internal interconnection of software systems using services relies on that existing, as well as new software systems, can be accessed as services. If only a minor part of internal and external systems can be interconnected using a service oriented approach, the full benefits of service orientation cannot be reached. The problem when moving from a point-wise use of services to a situation where services are used as the main infrastructure is that it becomes impossible to handle details of each individual service. For

example, when dealing with a small amount of services in point-to-point interconnections differences in functionality and technical specifications can be handled on a per-case basis. However, dealing with a large extent of services requires that the abstraction level is increased such that services can be combined on the business level, abstracting away from technical details. Moving to infrastructure use of services is thus hindered by the need to handle technical details, and the inability to treat services as modular resources on the business level.

An increased *reach* of service use relies on that services are constructed such that they are easily useable for external partners. The problem when moving from an internal to external use of services is that the services' functionality and their (possible) interactions with external partners must be readily available. For example, an internal service might follow implicit organizational-internal design principles on how to handle erroneous input messages, while an external service must make the handling of error situations explicit. This means that the way to use an external service should be clearly documented, and that the service should follow standardised protocols and conventions in the interactions with its external consumers. Thus, the move from internal to external services is hindered by the lack of standardised service descriptions that documents and enforces service capabilities and valid service interactions.

Reach and extent of service use can be used to broadly categorize the use of services into four different situations, or categories (Figure 1).

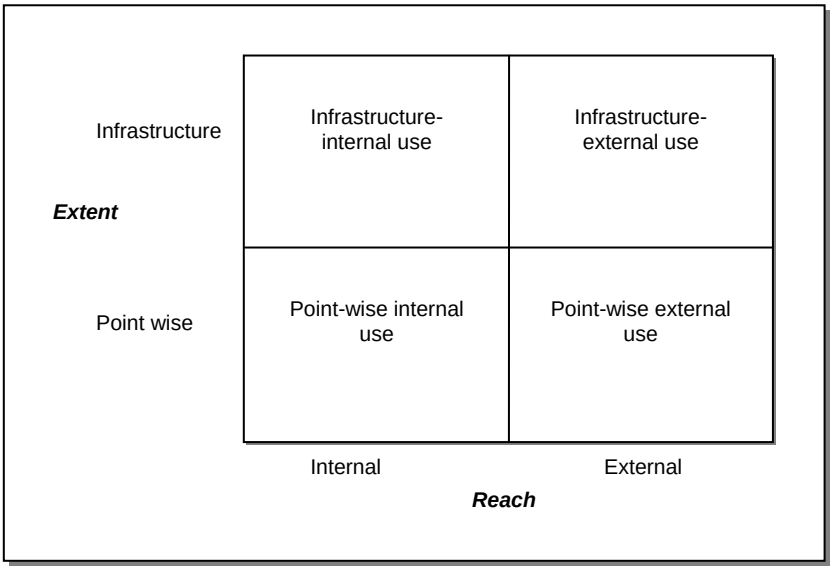

Figure 1, Categories of service use (from Paper A)

While point-wise internal use represents the least complex use of services, infrastructure-external use is the most complex situation. Interconnecting large amounts of services across organizational boundaries, for example in scientific calculations [12], would be an example of infrastructure-external use of services.

To achieve the full benefit of software services organizations must improve both the structure of existing internal systems, as well as enabling the systems to be connected to

external business partners. Thus, there is a need to improve both reach and extent of service use.

The problem is that increased reach and extent of service use is hindered by both the inability to handle services on an abstract level, as well as the lack of instruments for precise service interaction descriptions. Instruments to achieve reach and extent are thus fundamental for building service oriented systems. The contribution of this thesis is a set of such instruments.

3 Research Approach

The research approach consists of three main steps (Figure 2):

1) Defining service oriented computing, its goals and existing architectural instruments to reach these goals.
2) Defining a framework that relates the goals with service use categories, where each use category represents a typical use of services.
3) Providing instruments that support each of the defined use categories. These instruments can be generalized as either improving service oriented systems reach or extent, or both.

The first two parts, defining the goals and the high-level framework are reported in Paper A, while instruments for achieving reach and extent are presented in Papers B-D.

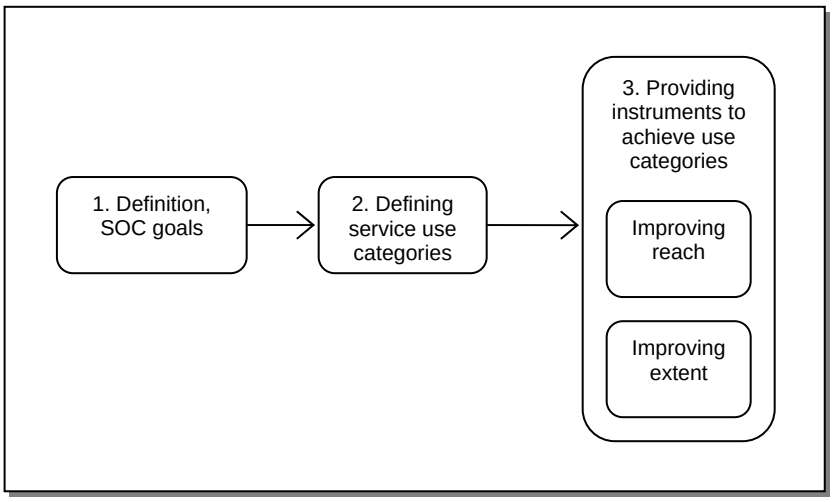

Figure 2, Steps in the research approach

The major part of the work has been in creating instruments for service oriented systems (Step 3), this work can be categorized as following a design science [13] approach. In design science the focus is on how to create novel artefacts that can be used in a domain, rather than the actual study and description of a domain. The study and description of a domain is rather the means to achieve a good design of the artefacts. In the information technology field the novel artefacts created by a design science approach can extend and improve applications of information technology [14]. The importance of design to improve information technology is, today, widely accepted [15]. Relating design science to this work

the artefacts are the presented instruments for increased reach and extent created in Step 3, while the presented framework is representing the study of the service oriented systems domain (Step 1-2).

The work presented in this thesis is grounded in both theory and practice. The framework constructed during step 1 and 2 provides a theoretical overview of the area of service oriented computing, describing its main concepts and structuring the problem domain of software services. The presented instruments to improve reach and extent is grounded in, and inspired by, a real world industrial case and case scenarios. Rooting the instruments in practical cases has the benefit of showing the need for, and the applicability of the instruments.

The provided instruments are based on two main concepts: The use of *processes* and *business – technology* integration. Processes are used as a means to enable the combination of services, while keeping a high abstraction level. Concepts for business – technology integration are used to describe the alignment between business issues, such as business processes, and concrete implementations of services.

Focusing on services in the context of processes and business - technology integration is not the only alternative when trying to improve the reach and extent of service oriented systems. Alternatives, as described briefly in Paper A, are to focus on architectural guidelines for modular services (thereby improving extent) and devising means to improve the collaboration of organizations by shared service repositories (improving reach). Providing high-level architectural guidelines largely follows what is well known for component based and object oriented systems, which can be seen in [16]. Investigation into service repositories mainly moves the research into the realms of service descriptions using concepts from semantic web [17][18] and the use of ontologies.

The benefit of basing instruments on the concepts of providing higher abstraction levels using processes and business-technology integration, is that the work contributes to improving both the reach and extent of systems. Furthermore, concentrating the work to a few fundamental concepts provides a coherent set of instruments, as well as, a starting point for further evolution of the instruments.

4 Contribution

The thesis presents a framework that can be used to guide the evolution and construction of service oriented system. Furthermore, the framework is used as the foundation for providing instruments that improves the reach (ability to handle large amounts of services) and extent (ability to handle cross-enterprise services) of architectures for software services.

As stated in the introduction, the contribution of the thesis is based on three main aspects; services as modules, technology and business perspectives of services and the combination of processes and services.

The instruments are positioned in three of the quadrants of the identified problem domain (see Figure 3).

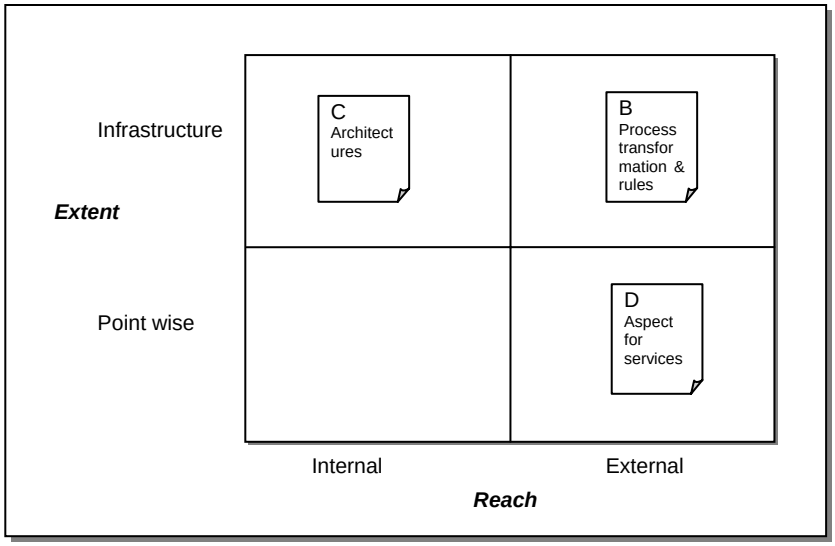

Figure 3, Overview of the framework and its relation to published papers.

The results are presented in the form of four published papers, all relating to different issues of service oriented computing;

A. A conceptual framework defining the scope of service oriented computing, what it can be used for, and the high-level architectural instruments that are needed to build a system based on software services.

B. A set of process transformation patterns that enables the use of combinations of services to be described at two abstraction levels, Business and Technical. Furthermore, guidelines to keep the abstraction levels synchronised are presented. The notion of business and technical processes is introduced to raise the abstraction level. This enables organizations to use high-level executable processes to interconnect services within and across organizations, thereby improving both *extent* and *reach*.

C. Based on the process transformation patterns, three high-level architectural styles that aid in the creation of modular service based systems are identified and analyzed. The main focus of the architectural patters is to enable the alignment of business and technology perspectives. The increased modularity due the use of the architectural styles improves the *extent* of service use.

D. The use of aspect oriented programming is proposed as a way to externalize previously organization-internal services. By applying aspects into existing services new business requirements put forward by external business partners can be satisfied. In service oriented computing, aspect oriented programming can thus be applied to improve the *reach*.

Paper A (the framework) is based on a literature survey, while paper B and C are based on an industrial case. Paper D is illustrated with a case scenario.

5 Related Work

Research in service oriented computing is a wide field spanning from low level implementation details to high level business scenarios. However, as this thesis focuses on business and technology integration and process abstractions in relation to services, related work can be pinpointed in those subfields.

Abstractions via processes

Existing proposals on how to introduce abstractions using process specifications can be classified into two categories. Firstly, some research suggests raising the abstraction level by the loss of details, thereby capturing a wider range of behaviour. Examples of constructs that raise the abstraction level are the use of ad-hoc sub-processes [19], activity inheritance [20], [21] and patterns of flexibility [22]. The use of two levels of processes (business and technical), as described in Paper B, fall into this category. The second category of process abstractions introduces constructs that are tailored to handle complex behaviour. These constructs are commonly "cross-cutting" abtractions, i.e. they control behaviour on a set of activities or an entire process. Examples of such constructs are Event-Condition-Action (ECA) rules [23], process parameterization [20] and the use of profiles to describe complex error handling [24]. The use of aspect oriented programming presented in Paper D belongs to this category of "cross-cutting" abstractions. Compared to related approaches, the overall approach presented in this thesis differs in that it views the realization of processes and services on two different levels – technical and business. While the approaches mentioned above view abstractions as constructs applied to a single process, the abstraction concept presented in this thesis is rather a relationship between two levels of processes (see Paper B).

Integration of business and technical perspectives

The problem of non-alignment between business and technical requirements in the design of enterprise systems has been widely studied in the research community ([25], [26], [27], [28], [29]). This thesis, however, specifically targets the problems related to service technology in that the foundation for the alignment is a set of system service constraints (Paper B). Moreover, this thesis specifically views the integration problem from a process-service perspective, thereby providing concrete solutions in the domain of service oriented computing (Paper C).

6 Summary of Included Papers

6.1 Paper A – General Framework

Paper A - Martin Henkel, *A Framework for Understanding the Vision, Goals, Instruments and Uses of Software Services*, Journal of Integrated Design and Process Science, Vol. 8, No. 1, pp 129-141, March 2004.

By introducing a framework that relates the vision and goals of software services to different categories of service use, this paper presents an overview of the instruments needed for the development of software services. The framework spans across four

interrelated parts; the vision, goals, instruments and categories of service use. Firstly a single vision for software services is proposed. This vision is then divided into the three goals of modularity, integration and discovery. Furthermore, the high-level instruments needed to achieve each goal are presented. The use of services is classified into four categories depending on reach and extent i.e.: point wise-internal, point wise-external, infrastructure-internal and infrastructure-external use. The framework presented in this paper shows how these categories of service utilise the defined instruments.

6.2 Paper B – Improving Extent and Reach by the use of Process Abstractions

Paper B - Martin Henkel, Jelena Zdravkovic, Paul Johannesson, *Service-Based Processes: Design for Business and Technology*, The 2nd International Conference on Service Oriented Computing (ICSOC'04), ACM Press, New York, USA, November 15-18, 2004.

In order to support business processes the design of the executable process must closely follow the business events and activities, as perceived by business actors. However, the design must also consider technical issues such as limitations in existing technology and systems. In this paper the influence of technical system constraints influence on the realization of business processes are examined. Based on this examination a set of realization types (process transformations) are presented that describes the transformation from a business process into its realization as an executable process. Furthermore, design criteria that need to be adhered to in order to cater to both business and technical needs are proposed.

6.3 Paper C – Improving Extent by using Architectural Styles

Paper C - Martin Henkel, Jelena Zdravkovic, *Architectures for Service-oriented Processes*, The Nordic Conference on Web Services (NCWS'04), Växsjö, Sweden, November 22-23, 2004.

In this paper we examine how the design of executable processes is affected by both technical and business issues. Furthermore, we examine a set of architectures that enable the use of executable processes to cater to both business and technical needs. Fundamental guidelines on how to apply the architectures are provided.

6.4 Paper D – Improving Reach by the use of Aspects

Paper D - Martin Henkel, Gustav Boström, Jaana Wäyrynen, *Moving from Internal to External Services using Aspects*, Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'05), Springer Verlag, Geneva, Switzerland, February 23 – 25, 2005.

In this paper we argue that aspect oriented programming is an important technique that can be used to facilitate the implementation of the new requirements that arises when moving from internal to external services. The suggested solution is illustrated by an example where quality of service metrics is implemented by using aspect oriented programming.

7 References

[1] Fremantle P., Weerawarana S., Khalaf R., "Enterprise Services," Communications of the ACM, Vol. 45, No 10, October 2002.

[2] Curbera F., Ferguson D., Nally M., Stockton M., "Toward a Programming Model for Service-Oriented Computing", ICSOC 2005, Third International Conference On Service Oriented Computing, Amsterdam, The Netherlands, December 12-15, 2005.

[3] Vinoski S., "Web Services Interaction Models, Part I: Current Practice", IEEE Internet Computing, Vol. 6, Iss. 3, pp. 89- 91, 2002.

[4] Curbera F., et. Al., "Business Process Execution Language for Web Services (BPEL4WS) v1.1," http://www.ibm.com/developerworks/library/ws-bpel/, May 2003.

[5] Chinnici R., et. Al., "Web Services Description Language (WSDL) 1.2", http://www.w3.org/TR/wsdl20/, March 2006.

[6] Gudgin M., et. Al., "SOAP Version 1.2", W3C Recommendation,June, http://www.w3.org/TR/soap12-part1, June 2003.

[7] Parnas L., "On the Criteria to be Used in Decomposing Systems Into Modules", CACM, December 1972.

[8] Cheesman J., Daniels J., "UML Components", ISBN 0-201-70851-5, Addison-Wesley, 2001.

[9] Carlsson B., Tyomkin D., "Service-orineted Architecture: Elements of Good Design", Journal of Business Integration, pp. 13-17, January 2004.

[10] Erl T., "Service Oriented Architecture – Concepts Technology and Design", Prentice Hall, ISBN 0-13-18-5858-0, 2005.

[11] Hull R., Benedikt M., Christophides V., J. Su. "Eservices: A look behind the curtain" In Proceedings of the International Symposium on Principles of Database Systems (PODS), ACM Press, San Diego CA, USA, June 2003.

[12] Foster I., "Service-Oriented Science", Science, Volume 308,6 May 2005.

[13] Simon H. A., "The Sciences of the Artificial", MIT Press, Cambridge, MA, 1996.

[14] Markus M. L., Majchrzak A., Gasser L., "A Design Theory for Systems that Support Emergent Knowledge Processes", MIS Quarterly, Vol 26, Iss 3, pp. 179-212, September 2002.

[15] Winograd T., "Bringing Design to Software", Addison-Wesley, ISBN 0-201-854910, 1996.

[16] Papazoglou M.P., Yang J. "Design Methodology for Web Services and Business Processes", Proceedings of the Technologies for E-Services Third International Workshop, TES 2002, Hong Kong, China, August 23-24, 2002.

[17] Berners-Lee T., Hendler J., Lassila O., "The Semantic Web", Scientific American, May 2001.

[18] Dumas M., O'Sullivan J., Heravizadeh M., Edmond D., Ter Hofstede A., "Towards a Semantic Framework for Service Description", Proceedings of the 9th IFIP Conference on Database Semantics, 2001.

[19] Heinl P., Horn S., Jablonski S., Neeb J., Stein K., Teschke M., "A Comprehensive Approach to Flexibility in Workflow Management Systems", WACC '99, USA, pp. 79-88, 1999.

[20] Aalst W.M.P. van der, "Flexible Workflow Management Systems: An Approach Based on Generic Process Models",. DEXA'99, LNCS 1677, pp. 186-195, 1999.

[21] Ribó J.M., Franch X., "Building Expressive and Flexible Process Models Using a UML-Based Approach", EWSPT 2001, LNCS 2077, pp. 152–172, 2001.

[22] Sadiq, Sh., Sadiq, W., Orlowska, M., "Pockets of Flexibility in Workflow Specifications", ER2001, LNCS 2224, Yokohama, Japan, pp. 513-526, November 2001.

[23] Joeris G., Herzog O., "Towards Flexible and High-Level Modeling and Enacting of Processes", CAiSE'99, LNCS 1626, pp. 88-102, 1999.

[24] Chopra A., Singh M., "Commitments for Flexible Business Processes", AAMAS'04, July 19-23, New York, USA, 2004.

[25] Bubenko, J.A., Jr., Wangler, B., "Objective driven capture of business rules and of information systems requirements", Proceedings of the IEEE Systems Man and Cybernetics Conference, France, Le Touquet, pp. 670-677, October 1993.

[26] Grover V., Fiedler K., Teng J.T.C.: IEEE Transactions on Engineering Management, Vol 41, Iss 3, pp. 276 – 284, August 1994.

[27] Yu E. S. K., Du Bois P., Dubois E., Mylopoulos J., "From Organization Models to System Requirements, A 'Cooperating Agents' Approach", Proc. of the Third International Conf. on Cooperative Information Systems (CoopIS'95), Vienna , pp. 194-204, May 1995.

[28] Rolland C., Prakash N., "Bridging the Gap Between Organisational Needs and ERP Functionality", Journal of Requirements Engineering, Vol 5, Iss 3, pp. 180-193, 2000.

[29] Rolland C., Prakash N., "Matching ERP System Functionality to Customer Requirements", 5th IEEE International Symposium on Requirements Engineering (RE 2001), Canada, Toronto, IEEE Computer Society, pp. 66-75, August 2001.

Paper A

A Framework for Understanding the Vision, Goals, Instruments and Uses of Software Services

Martin Henkel

Journal of Integrated Design and Process Science, Vol. 8, No. 1, pp 129-141, March 2004.

A FRAMEWORK FOR UNDERSTANDING THE VISION, GOALS, INSTRUMENTS AND USES OF SOFTWARE SERVICES

Martin Henkel
Department of Computer and Systems Sciences, Stockholm University and Royal Institute of Technology, Stockholm, Sweden

Software services have been suggested for use in several areas such as business-to-business communication, system integration and as an enabler for virtual enterprises. However, it has not been clear if the use of services in these different areas is guided by a common vision, and if they share goals and technical background. By introducing a framework that relates the vision and goals of software services to different categories of service use, this work presents an overview of the instruments needed for the development of software services. The framework spans across four interrelated parts; the vision, goals, instruments and categories of service use. Firstly a single vision for software services is proposed. This vision is then divided into the three goals of modularity, integration and discovery. Furthermore, the high-level instruments needed to achieve each goal are presented. The use of services is classified into four categories depending on reach and extent i.e.: point wise-internal, point wise-external, infrastructure-internal and infrastructure-external use. The framework presented in this paper shows how these categories of service utilise the defined instruments. As an example of framework use, the instruments needed to create a process-centric service infrastructure for intra-enterprise use is proposed

Keywords: *Service oriented computing, e-services, software services*

1. Introduction

The terms service, e-service and Web service have begun to be promoted in product launches as well as academic and industrial research. Standards have been agreed upon to define service technologies such as SOAP, WSDL and UDDI (Gudin *et al.*, 2002; Chinnici *et al.*, 2003, Bellwood *et al.*, 2002). Drafts of standards are quickly incorporated in new and existing products. Research has extended the service domain with additions such as repositories and agent-based service brokers. Clearly, services are drawing a lot of attention.

The vision driving the development of software-services is somewhat ambiguous and not always obvious. The vendors of Enterprise Application Integration (EAI) servers tout the use of services as a way to reuse and connect existing applications. In Business-to-Business communication (B2B) services are used as enablers of a global, open e-market. Identifying and understanding the vision of software services is important as a first step towards identifying what is required to enable the application of software-services.

Starting with the vision of software services this paper identifies three basic goals for software services. These goals are in effect drivers for the development of new products as well as being the impetus for initiating research into services. To achieve these goals, a set of instruments can be utilised.

Instruments can be both of technical nature such as a standard technical infrastructure, or of a more prescriptive nature such as the different ways to decompose and compose software services. The instruments can be used to solve common problems, such as providing a way to modularise a system based on services. The instruments required to fulfil each of the three goals are identified and described in this paper.

When using software services, one can strive towards one or more of the identified goals. Striving towards several of the defined goals demands the utilisation of several instruments. By classifying uses of services into categories it is possible to propose a set of instruments that, for each category, is suitable for goal fulfilment.

By identifying vision, goals, instruments, categories of uses and their interrelation it is possible to get a structured overview of the uses of software services, along with a set of instruments that are required to successfully use software services. An overview of the parts in the framework is presented in Figure 1.

The paper is structured as follows: In the first section the term service is defined and services are put in context by comparing services and components. In the next section a vision for software services is proposed. This vision is broad by purpose, so that it encompasses current research issues and industry trends. In the following section the vision is divided into three goals, each characterizing a typical issue within the field of software services. Then, the instruments that are needed to fulfil these goals are presented. These instruments represent a range of differentiated solutions from both academic research and existing commercial products. In the next section uses of services are categorized, the chosen criteria of categorization are situations of service use that greatly impact the need for instruments. Furthermore, each use category is related to the defined goals and instruments. The paper ends with an example of how to use the framework.

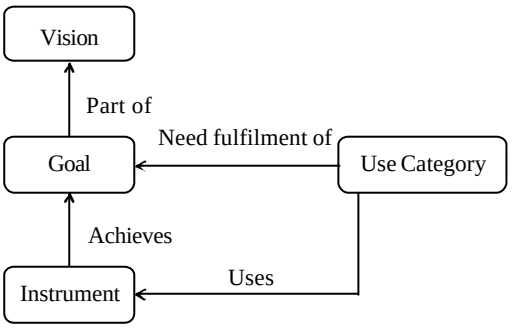

Fig. 1 Overview of the framework

2. Definitions

Disregarding the software domain, a service can be defined as an "act or performance offered by one party to another" (Lovelock *et al.*, 1996). From a business perspective, a service can be viewed as a process that produces value for the consumer of the service. The parties involved in this definition are not limited to software, i.e. a party can be either human or software. Regardless of whether a service is used and provided by humans or software, a service does not result in any ownership (Bennett *et al.*, 2001b; Dumas *et al*., 2001). This means that a service is an offering that can be used, compared to a commodity that can be owned and bought.

An application service provider (ASP) is an example of a service where the provider is a software system, and the consumer is a human.

In this paper, the term *software* service is used to denote a service that can be used by software systems, or speaking in software terms, a service can be invoked from other software systems. In this definition both parties are software programs that communicate, potentially over a network. Just as for non-software services the consumer and provider of a software service can belong to separate enterprises. The remainder of this paper will focus on software services.

2.1. Services and Components

Services share many properties with components. Firstly, both services and components offer a well-defined interface that hides the details of implementation. Secondly, both services and components are modular so that they can be combined to form new components and services, respectively. The important difference is that a service is a run-time entity offering a service interface while a component is a physical/binary entity that needs to be installed before use. Users of components need to buy them, install them and then use them, while a service user simply finds the service and uses it (Herzum, 2001). This difference implies that the provider of a service is responsible for it, even in run-time, while the provider of a component does not necessarily have any responsibility and control of the run-time environment.

Component technologies like COM+ (Platt, 1999) and EJB (Monson-Haefel, 2001) are useful for building software services. An EJB component for example, installed and running on a server can be offered as a service. It can be said that components are developed, but they should be provided as services (Allen and Frost 1998). This means that components are primarily used as building blocks in the development phase, whereas services exist in run-time.

Not all components of a system should be provided as services since the user of an application (software or human) is seldom interested in the inner workings of the system. A black-box view is preferred. By selecting the components to be provided as services, an uncluttered interface to the system can be obtained. Thus, the exposed services are commonly (but not necessarily) more coarsely grained than the components that together implement the service.

Due to the run-time properties of services, it is natural to focus on the service interface and on run-time communication when discussing services. When one on the other hand is discussing components, the development, deployment and run-time environments come into focus.

There are then four properties of components and services that can be compared. The comparison based on these properties is summarized in Table 1.

Table 1 Components and services compared

Property	Components	Services
Use	Buy-install-use	Use-pay
Responsibility of provider	Construction and delivery	Run-time service availability
Granularity	Application building block	Application interface
Focus of interest	Implementation	Communication

3. Service Vision

A long-term vision behind software services is to *enable a global market of well-structured, well-defined and modular services that are accessible for consumers around the world*. In this vision, services are highly modular resources that can be composed to form new systems and services (Piccinelli and Stammers, 2001; Péraire and Coleman, 2000; Bennet *et al.*, 2001b). Furthermore, an important part of the vision is that consumers should be able to select and combine services offered by providers that might reside on different geographical locations and be provided by different organisations. The vision includes that services should be platform and program language independent (Fremantle *et al.*, 2002).

By using services it is possible for organisations to out-source complex functionality to other organisations (Harumi, 2000), thereby forming virtual organisations or "networked enterprises" (Yang, 2001).

4. Service Goals

The vision of software services can be divided into sub-visions, or goals. Firstly, the functionality needs to be divided into modular resources. Secondly, these resources must use an interconnection format that makes them globally useable. Thirdly, consumers must be able to discover and select the service to use.

Thus, the vision for software services can be divided into three goals; modular resources, global interfaces and dynamic discovery.

Modular resources: This goal is to provide each resource as a service, instead of encapsulating and tying them into monolithic applications. The major benefit of creating modular services is that new systems can be built by combining existing services. This makes it easier to implement changes derived from evolving requirements. A typical example is the development of new business processes that need IT support. The notion of services as building blocks for systems presented by Wald and Stammers (2001) is an example of how to benefit from services as modular resources.

Global interfaces: The goal of services as global interfaces means that services should define and be used as a standard for connecting software systems. This goal includes removing interconnection boundaries such as differences in platform, program languages, and geographical location. As a further step the differences in the way organisations do business that hinder communication can be overcome by defining a standard way to handle business agreements for services. Standardisation and description of interfaces and protocols are essential requirements of this goal. Work being conducted on ebXML and Web service technologies is an example of steps towards achieving this goal (Webber and Kotok, 2001).

Dynamic discovery: At least a partial fulfilment of the goal of modular resources and global interfaces are fundamental for creating a global market of well-structured, well-defined services that are accessible for consumers around the world. The goal of dynamic discovery of services consists of this market, together with consumers who can actively search the market for suitable services offered by providers. Consumers can select the provider who offers the best price, performance etc, etc. By constantly searching the market for new providers the consumer can change provider, giving optimal performance at every moment. The architecture for dynamic evolving systems based on services presented by Bennet *et al.* (2001a) is an example of the implementation of this goal.

5. Service Instruments

To achieve the defined goals, development and provisioning of services can utilise a set of instruments. The following sections state the requirements for each goal and describe a set of instruments that can be utilised to achieve each goal.

5.1. Instruments for Modular Resources

Representing a major part of an organisation's resources as modular services requires that the functionality can be decomposed into services. Decomposition into services should be done in a way that enables consumers to freely combine services, without the risk of duplicating functionality, and without the need for excessive customisation to make the services fit together. Constructing a set of services that are meant to be combined requires stringent decomposition criteria. A set of high-level criteria for decomposition is thus valuable instruments when striving for the goal of modular resources.

A major criterion for decomposition is if the services should be aimed at either supporting a technical aspect or an organisational aspect. Technical aspects include (database) transaction handling, queues and events, persistence etc. Organisational aspects include business processes and tasks. A service built to support a technical aspect is called a horizontal service (Stahl, 2002), whereas services that support organisational aspects are called vertical services. Using vertical and horizontal decompositions is a useful instrument for the development of modular services.

Horizontal services are independent on the business domain. The main benefit of horizontal services is that they can be used by several applications in different domains, thereby providing a technical framework for system development. Common horizontal services include error and transaction handling, asynchronous event handling and security (Brown, 2000). Horizontal services such as transaction co-ordination and events are commonly provided by middleware products such as COM+ and products implementing CORBA (Orfali *et al.*, 1996). When developing an application the decision can be made to simultaneously develop horizontal services that support the entire application. The horizontal services then become an important part of the technical infrastructure in the application (Herzum and Sims 2000).

Vertical services are constructed in a context of a business domain. The criterion for structuring functionality into vertical services is that the service should represent a part of the organisation, or a need of the organisation. Two examples of decomposition are letting the services represent business functions (Wald and Stammers 2001), or letting the service interface represent a business process. A vertical decomposition of services is useful when using services as means of business process integration. Vertical services are equally important in inter and intra organisational integration.

Decomposition into vertical and horizontal services can be used in combination to utilise the advantages of both approaches.

5.2. Instruments for Global Interfaces

The use of services as a way to interconnect possibly heterogeneous systems requires the use of an "interconnection stack" ranging from technical network protocols and interface descriptions to semantic descriptions of how to use the service. Defining and standardising the levels in such an interconnection stack give consumers and providers a common means for communication. To provide a solution for communication differences on each level in the stack is a vital instrument to reaching the goal of services as global interfaces.

The interconnection stack can be divided into three distinct levels. The first level, technical infrastructure, consists of the basic "plumbing" needed for services to communicate, regardless of business domain and provider. The second layer, interface descriptions, is needed to communicate with a specific service. The third level, the semantic level, is needed to understand the context of service use. The levels are described further in the following paragraphs.

Technical infrastructure descriptions define how to communicate with the service. These descriptions include which communication protocols the service can use, such as TCP/IP or UDP. Technical descriptions also define the use of higher-level communication protocols such as IIOP (Orfali *et al.*, 1996), SOAP and RMI (Monson-Haefel, 2001). In brief, the technical description is sufficient for the consumer to connect to the service.

Interface descriptions define the programmatic interface of the services, that is, the methods that the service exposes. The description includes method names, parameters and data structures needed to call the service. These descriptions are commonly done with an interface definition language (IDL). An example of such a language is the Web Service Description Language (WSDL). In conjunction with the technical infrastructure the interface description makes it technically possible for a consumer to connect and call the service.

Semantic descriptions contain the context in which the service can be used, including the meaning of the methods in the service interface. A simple example of a semantic description for the method "add(x,y)" is that the method adds two numbers and returns the result. The consumers can certainly call the method without knowing the semantics, but they can never use it in a meaningful way without the semantic description. Semantic descriptions may include a set of conditions for use, for example by referring to standardised business contracts. Meta-data description languages such as the Ontology Web Language (OWL) can be used to extend the interface descriptions with descriptions at the semantic level (Smith *et al.*, 2002). Another example of high-level descriptions of services is process descriptions that define how a set of services can be combined to form a process. An example of a language for process description is the Business Process Execution Language for Web Services, BPEL4WS (Curbera *et al.*, 2002). BPEL4WS or other process description languages can be used to describe typical interaction schemes for consumers and providers. These interaction descriptions can be seen as a form of semantic description of the context of service use.

All of the above service descriptions are necessary in order to use a service. However, the descriptions need not be defined in an explicit way or in a way that is interpretable by software. Making service descriptions interpretable by software systems allows consumers to examine the service and to adapt to changes in technical infrastructure, interfaces and semantics.

5.3. Instruments for Dynamic Discovery

The goal of dynamic discovery is that a service consumer can switch between providers offering suitable services. In order to be of any benefit for the consumer, dynamic discovery requires a market of well-structured services.

Dynamic discovery requires that the consumers can perform searches for suitable services. During the search the consumer needs are compared to existing services offered by providers. The process of matching the consumer needs with the offered services is a form of brokering, or "matchmaking". Brokering is dependent on a comparable description of both the consumer needs and of the provided services. Description of consumer needs and services can be done on the technical, interface and semantic levels.

Brokering can be done by allowing the providers to register their service descriptions in a registry. A broker that matches the requirements with the contents of the registry can utilise different approaches to the brokering process. Besides the approach of the broker, the consumer's way of using the broker can also impact the outcome of dynamic service discovery.

Approaches to brokering, and different use of brokering are important instruments when implementing dynamic discovery. Approaches to brokering and consumer's use of brokering are described in the following sections.

5.3.1. Approaches to Brokering

The ability of the broker to find the requested service is dependent on the content of the registry and on the matching algorithms applied to find the service. Given a request from the consumer the broker searches its registry to find an appropriate provider. The brokering algorithm can be divided into three basic types; frame-based, specification-based and brokering based on mediator schemes.

Frame-based brokering is a way to search the service descriptions for values of certain fixed attributes such as name and type of the service (Klein and Bernstein, 2001). Both the service description and the query for services can be described as collections, or frames of attribute-value pairs. An example of brokers that are frame-based is brokers that implement the Universal Discovery, Description and Integration (UDDI) standard.

Specification brokering, or deductive brokering is an extension to frame-based brokering. By providing the broker with a set of rules relating to how to interpret service descriptions, the broker can find services by applying the rules to the service descriptions. An example of specification brokering is if the broker can deal with generalisations. For example, a service that provides global packet deliveries is a generalisation of a local packet delivery service. When a consumer searches for a local packet delivery service, the broker can find and return both local and global delivery services by applying the rule of generalisation. Specification brokering can also be done at the method level, which has been proposed as an aid in searching software repositories (Rollings and Wing, 1991). Compared to the frame-based brokering approach the specification-based approach requires a detailed description of the services on the interface level, and a description meta-model on the semantic level.

Mediator schemes for brokering lets both the provider and consumer actively participate in the brokering process. The advantage of using a mediator scheme is that the provider and/or the requestor can actively participate in the brokering process, thereby extending the brokering algorithm. Active participation can be achieved by using agents representing the provider and/or the requestor. An example of a broker with agents representing the providers is presented by Helal *et al.* (2001).

5.3.2. Consumer Use of Brokering

The broker contains the brokering algorithm and a registry of the providers. Consumers can control the outcome of the brokering process by altering the service request. Another important aspect that the consumer can affect is the binding time.

Bind time affects how often the consumer searches for a provider. If the search for providers is performed for each time the service is needed, the consumer can find a new provider as soon as it is available. This kind of binding is called ultra-late-binding (Bennett *et al.*, 2000) or *just-in-time binding* (JIT) (Andrade and Fiadeiro, 2001). JIT binding is not always feasible since it is associated with a performance overhead. Instead of finding a new provider for each call the requestor can bind to a provider for the duration of a session. A single session can span across multiple calls, thereby alleviating the overhead of JIT binding. Yet another approach is to use the same provider in a fixed time-frame. The fixed time-frame can be chosen to minimise the performance penalties, or be stated in a contract between the provider and the consumer.

Table 2 summarises the goals and the high-level instruments that can be used to achieve the goals.

Table 2 Goals and instruments

Goal	Instruments
Modular resources	The use of vertical and/or horizontal decomposition. Let services represent business functions or business processes. Use horizontal services for transactions, events and security.
Global Interfaces	Describe and standardise the technical infrastructure, interface and semantic aspects of provided services.
Dynamic discovery	Usage of frame, specification or mediator/agent based brokers. The use of fixed, session or JIT binding.

6. Applying Services

Users of services need not embrace all of the defined goals. Instead, partially fulfilling the goals can be enough in certain situations. Fulfilling part of the goals can be done by selecting the instruments to be utilised. By selecting and combining instruments, service-based systems can be tailored to a certain need. For example, a fixed point-to-point integration between two systems can initially use a loosely-defined vertical-service decomposition (an instrument to achieve the goal of modular systems) with a well-defined communication standard on the technical and interface level (an instrument to achieve the goal of global interfaces). Later, when the need to integrate more systems arises, the solution can be extended by using more instruments such as a simple frame-based broker (part of the dynamic-discovery goal).

By examining the situation where services are to be used, it is possible to identify which service goals to strive for. Then, using the defined goals it is possible to select the instruments that is best suited for the given situation.

The next section presents a categorisation of service uses. For each category, the service goals and instruments that are suitable for application are identified.

6.1. Categories of Service Use

Uses of services can be examined along two dimensions, the extent of service use, and the reach of service use. How extensive service use is, is determined by deciding if services are used as the key infrastructure/architecture for system construction, or if services are used for solving single, point-wise problems. The reach of service use can be divided into use within an organisation and use between organisations.

The differentiation between infrastructure use and point-wise use of services is important as it impacts on the need for precise decomposition and standardised service descriptions. Using a small set of services as point-wise solutions does not require precise decomposition criteria and interconnection standards. The small amount of services makes it possible to handle differences in composition and interconnection formats on a case-to-case basis. However, managing a large set of services with different communication standards and decompositions is expensive. Hence, if services are to be used as the main infrastructure, the importance of service decomposition and descriptions is increased. To summarise, using services as the main infrastructure/architecture relies on fulfilment of the goal of modular resources and partial fulfilment of the goal of global interfaces.

The difference between using services within an organisation and between organisations affects the need to have precise service descriptions and the need to utilise service brokers. Communication within an organisation can rely on implicit and partially-specialised protocols. When communicating with several organisations, the importance of standardized agreements such as global communication standards and standard business contracts becomes increasingly important. Utilising a service broker introduces a layer of indirection between the organisations, allowing the organisations to modify the service implementation and location without affecting the other party. Hence, using services to communicate between organisations relies on partial fulfilment of the goals of global interfaces and dynamic discovery.

Combining the dimensions of reach and service-use extent gives four combinations of service use; point-wise-internal, infrastructure-internal, point-wise-external, and infrastructure-external. Figure 2 summarises how the four uses of services are aided by fulfilment of the three service goals In this classification of service use each of the two dimensions, reach and extent, is divided into two areas of service use. This rough division is used to highlight differences in service use. However, examining an entire organisation's use of services might very well require a more detailed grading of the two dimensions.

Each of the four categories of service use and which instruments they utilise are described in more detail in the following section.

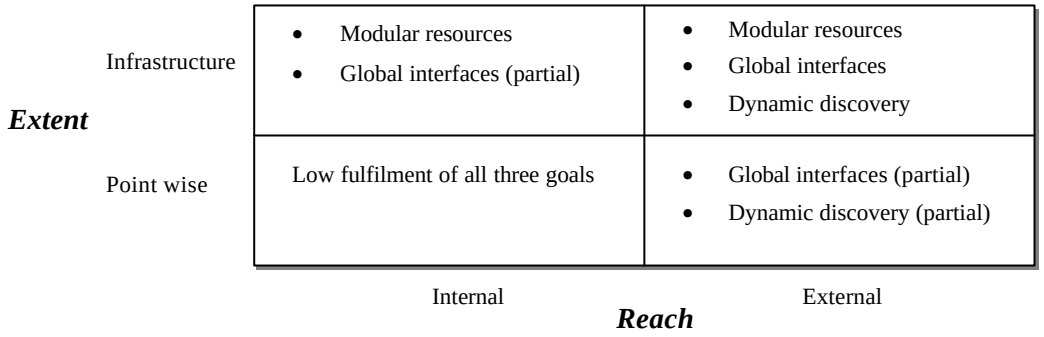

Fig. 2 The service use categories and their required goal fulfilment

6.2. Service Use and Instruments

A ***Point-wise-internal*** use of services is when services are used for single solutions within an organisation, such as when integrating existing legacy systems. Since the use of services is sparse, the need for precise service decomposition rules is low. The use of standardised technical infrastructures can help development, but since the use is internal, non-standardised solutions can be used. This kind of service use can be classified as enterprise application integration (EAI) with services.

An ***Infrastructure-internal*** use of services is when services are used as the key architecture for system development within an organisation. Using clear service decompositions, by, for example, vertically structuring the services according to business function makes the services easier to combine. As the amount of services grows, the need for standardised descriptions on the technical and interface levels increase. Use of services as the main architecture for structuring internal resources will create an internal collection of ready-to-use services, this approach combined with a message-based architecture is called an enterprise service bus (ESB) (Chappel, 2002).

A ***Point-wise-external*** use of services is when services are used for single point application integration between two organisations. Compared to point-wise-internal use, point-wise-external use is categorised by its increased need for standardised technical infrastructures and interface descriptions. Agreeing upon a non-standardised infrastructure can be difficult, since the two organisations might use different techniques internally. Using a frame-based broker with fixed or session binding alleviates the need for a tight coupling between single machines on the consumer and provider sides. This use of services as a point-wise bridge between organisations can be seen as a static business-to-business (B2B) connection.

An ***Infrastructure-external*** use of services is when services are used to connect multiple organisations in an automated way. Using services as an external infrastructure enables service consumers to find services on a global service market. Finding services is made possible by utilising service brokers. Using specification-based or mediator/agent-based brokering combined with JIT or session binding enables consumers to find and change providers. Since consumers can communicate with many providers, it is important to use highly standardised service descriptions and technical infrastructures. The use of service descriptions at the semantic level enables automatic negotiation of contracts for service use. Use of services as a global, dynamic interconnection for business can be seen as a form of dynamic business-to-business (B2B) communication.

The four categories of service-use and their required instruments are summarized in Figure 3.

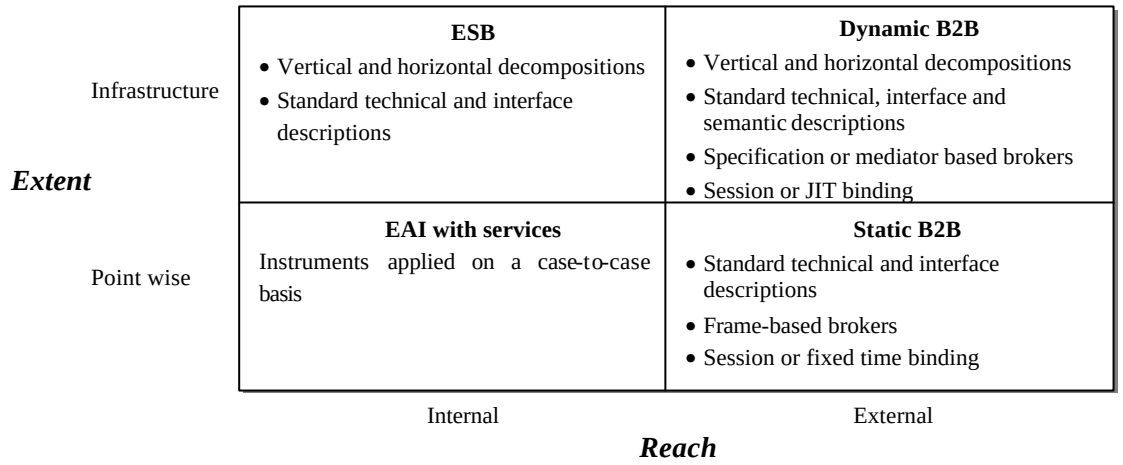

Fig. 3 The service use categories and needed instruments

7. Applying the Framework

The framework presented in the previous sections can be used as an aid for selecting suitable instruments when developing services. The following four-step process can be used as a guideline:

(1) Identify the problem domain.
(2) Map the problem domain to one of the four categories of use, as presented in the framework.
(3) Use the framework for selecting the instruments that need to be used.
(4) Decide how to implement the required instruments.

The next section presents an example of framework use. The example shows how to use the framework to guide the creation a process-centric infrastructure for enterprise use.

7.1. A Process Centric Infrastructure

A problem that organisations face when moving towards a process-centric structure is that current IT solutions are organised in a stovepipe-like architecture where each application supports a single department within the organisation. The result is isolated applications that are problematic to integrate (Johannesson and Perjons, 2000). A process-centric organisation requires that all of the applications in the organisation can be integrated and combined to support new business processes.

By following the four-step guideline previously outlined, it is possible to identify the category of use and select the appropriate instruments needed when developing services for a process-centric organisation. The following four paragraphs describe how to follow the guidelines:

(1) *Identify the problem domain*. In this case the problem is to support a process-centric organisation with services that can be easily combined and extended to support new business processes.

(2) *Map problem domain to use category.* Firstly, the described problem involves only resources internal to the organisation. Secondly, the wish to be able to combine a potentially large set of services suggests that services should be the main way of representing the business IT support. The category of service use that resembles these requirements is infrastructure-internal use of services.

(3) *Select instruments.* By using the framework, it can be concluded that infrastructure-internal use of services requires a well-defined decomposition criteria, along with the use of standardised service

descriptions on the technical infrastructure and interface levels. However, the need for advanced brokering is not an issue in this situation since the services are for intra-organisational use only. Thus the instruments needed in this situation are well-defined decomposition criteria, as well as service description and standards on the technical infrastructure and interface level.

(4) *Decide how to implement the instruments.* When the required instruments are selected, it is first necessary to select a criterion for decomposition. The organisation in this situation is process-centric, so that a suitable decomposition criterion might be a vertical decomposition where each service provides support for a single business process. An additional layer of services can be organised to support the underlying business functions (another vertical decomposition). After selecting the decomposition criterion a suitable technical infrastructure needs to be selected. Several vendors provide both standardised and proprietary technical infrastructures. An example of a suitable technical infrastructure would be Web-services, using SOAP on top of HTTP as the means for communication. Using Web-services as the infrastructure, a suitable interface description language would be WSDL.

The suggested solution can be extended with the use of instruments for service descriptions on the semantic level, such as OWL. Furthermore, the solution can be augmented with an UDDI compliant broker for intra-organisational use.

8. Conclusion and Future Work

This paper has presented a vision behind software services, and identified three distinct goals that are a prerequisite for achieving this vision. The instruments required for achieving each of these goals were presented. Furthermore, a classification of service uses was presented, and it was shown how different categories of service usage could take advantage of the defined instruments.

These elements have been combined to form a framework for the interconnection of service vision, goals, instruments and uses. This framework gives us a context for discussing, developing and using software services.

The potential use of the framework has been demonstrated by proposing a set of instruments needed to create process-centric services for intra-enterprise use.

The presented framework is a coarse-grained overview of the vision, goals, instruments and uses of software services. Future work will entail the extension of the framework both in detail and scope. Two possible extensions to the detail of the framework are to develop guidelines for selecting and combining decomposition criteria, and to further examine the situations where advanced brokering is suitable. The scope of the framework can be extended by including a mapping of the described high-level instruments to currently available technologies such as Web services, ontologies and ebXML. The scope could also be extended by mapping the use categories to requirements on non-functional aspects such as security and quality.

9. References

Allen, P., and Frost, S., 1998, “Component-Based Development for Enterprise Systems: Applying the Select Perspective,” Cambridge University Press.

Andrade, L., and Fiadeiro, J., 2001, ”Coordination Technologies for Web-Services,” OOPSLA Workshop on Object-Oriented Web Services.

Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y. L., Januszewski, K., Lee, S., McKee, B., Munter, J., and von Riegen, C., 2002, "UDDI Version 3.0," Published Specification, 19 July 2002, http://uddi.org/pubs/uddi_v3.htm, Accessed 7 Jan 2003.

Bennett, K., Layzell, P., Budgen, D., Brereton, P., Macaulay, L., and Munro, M., 2000, “Service-based software: the future for flexible software,” Proceedings of the 7th Asia-Pacific Conference on Software Engineering, APSEC 2000, pp. 214-221.

Bennet, K., Munro, M., Gold, N., Layzell, P., Budgen, D., and Brereton, P., 2001a, "An Architectural Model for Service-Based Software with Ultra Rapid Evolution," Proceedings of IEEE International Conference on Software Maintenance, pp. 292 – 300.

Bennett, K., Jie Xu, Munro, M., Zhuang Hong, Layzell, P., Gold, N., Budgen, D., and Brereton, P., 2001b, "An architectural model for service-based flexible software," 25th Annual International Conference on Computer Software and Applications, COMPSAC, pp. 137 -142.

Brown, A. W., 2000, "Large-Scale Component-based Development," Prentice Hall Inc.

Chappell, D., 2002, "Asynchronous Web Services and the Enterprise Service Bus," http://www.webservices.org/index.php/article/articleview/352/4/24/, Accessed 7 Jan 2003.

Chinnici, R., Gudin, M., Moreau, J., and Weerawarana, S., 2003, "Web Services Description Language (WSDL) Version 1.2," W3C Working Draft 24 January 2003.

Curbera, F., Goland, Y., Klein, J., Leymann, F., Roller, D., Thatte, S., and Weerawarana, S., 2002, "Business Process Execution Language for Web Services, Version 1.0," Public draft release 31 Jul 2002, http://www-106.ibm.com/developerworks/library/ws-bpel, Accessed 3 Feb 2003.

Dumas, M., O'Sullivan, J. , Heravizadeh, M., Edmond, D., and Ter Hofstede, A., 2001, "Towards a Semantic Framework for Service Description," Proceedings of the 9th IFIP Conference on Database Semantics.

Gudin, M., Hadley, M., Mendelsohn, N., Moreau, J., and Nielsen, H. F., 2002, "SOAP Version 1.2," W3C Candidate Recommendation 19 December 2002.

Fremantle, P., Weerawarana, S., and Khalaf, R., 2002, "Enterprise Services," Communications of the ACM, October 2002, Vol. 45, No 10.

Harumi, K., 2000, "Surveying the E-Services Technical Landscape," Second International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems, pp. 94-101.

Helal, S., Wang, M., Jagatheesan, A., and Krithivasan, R., 2001, "Brokering based self organizing e-service communities," Proceedings of the 5th International Symposium on Autonomous Decentralized Systems, pp. 349-356.

Herzum, P., 2001, "Web Services and Service- Oriented Architectures," Distributed Enterprise Architecture Advisory Service, Executive Report Vol. 4, No. 10.

Herzum, P., and Sims, O., 2000, "Business Component Factory," OMG Press.

Johannesson, P., and Perjons, E., 2000, "Design Principles for Application Integration," 12th Conference on Advanced Information Systems Engineering, Springer LNCS.

Klein, M., and Bernstein, A., 2001, "Searching for Services on the Semantic Web Using Process Ontologies," Proceedings of the Semantic Web Working Symposium.

Lovelock, C., Vandermerwe, S., and Lewis, B., 1996, "Services Marketing," Prentice Hall Europe.

Monson-Haefel, R., 2001, "Enterprise JavaBeans," O'Reilly & Associates.

Orfali, R., Harkey, D., and Edwards, J., 1996, "The Essential Distributed Objects Survival Guide," John Wiley & Sons Inc.

Péraire, C., and Coleman, D., 2000, "Modeling for E-Service Creation," Technical Report, SRI international.

Piccinelli, G., and Stammers, E., 2001, "From E-Process to E-Networks: and E-Service-oriented approach," OOPSLA Workshop on Object-Oriented Web Services.

Platt, D. S, 1999, "Understanding COM+," Microsoft Press.

Rollings, E., and Wing, J., 1991, "Specifications as Search Keys for Software Libraries," Proceedings of the 8th International Conference on Logic Programming.

Smith, M. K., McGuinness D., Volz, R., and Welty, C., 2002, "Web Ontology Language (OWL) Guide Version 1.0," W3C Working Draft 4 November 2002.

Stahl, M. , 2002, "Beyond Component-Based Computing," Communications of the ACM, October 2002, Vol. 45, No 10.

Wald, E., and Stammers, E., 2001, "Out of the Alligator Pool: A Service-Oriented Approach to Application Development," EAI Journal, March 2001.

Webber, D., and Kotok, A., 2001, "ebXML: The New Global Standard for Doing Business on the Internet," New Riders Publishing.

Yang , J., van den Heuvel, W. J., and Papazoglou, M. P., 2001, "Service Deployment for Virtual Enterprises," Australian Computer Science Communications, Vol. 23, Iss. 6.

Paper B

Service-based Processes - Design for Business and Technology

Martin Henkel, Jelena Zdravkovic and Paul Johannesson

Proceedings of the 2nd International Conference on Service Oriented Computing (ICSOC'04), ACM Press, New York, USA, November 15-18, 2004.

Service-based Processes – Design for Business and Technology

Martin Henkel
Stockholm University and
Royal Institute of Technology
Forum 100, 164 40 Kista
Sweden
+46 8 16 16 42
martinh@dsv.su.se

Jelena Zdravkovic
University of Gävle and
Royal Institute of Technology
801 76 Gävle
Sweden
+46 26 64 83 06
jzc@dsv.su.se

Paul Johannesson
Stockholm University and
Royal Institute of Technology
Forum 100, 164 40 Kista
Sweden
+46 8 16 16 71
pajo@dsv.su.se

ABSTRACT
Composition of software services is a fundamental part in supporting enterprise business processes. Designed properly, executable processes can be used to closely support business processes by the integration of existing software services. In order to support business processes the design of the executable process must closely follow the business events and activities, as perceived by business actors. However, the design must also consider technical issues such as limitations in existing technology and systems. In this paper we examine how technical system constraints influence the realization of business processes. Based on this examination we present a set of realization types that describes the transformation from a business process into its realization as an executable process. We also propose design criteria that need to be adhered to in order to cater to both business and technical needs.

Categories and Subject Descriptors
H.1.0 [**Models and Principles**]: General*;*
H.4.1 [**Information Systems Applications**]: Office Automation – *Workflow management*

General Terms
Design, Languages. Theory.

Keywords
Business processes, Executable processes, Service coordination.

1. INTRODUCTION
Large scale service-oriented systems rely on complex message exchanges between individual software services. As the complexity and the number of interactions increases, there is a need to explicitly control and implement the service interactions. Executable processes enable individual services to be composed to support new, complex business interactions [1, 2].

When designing executable processes, consideration must be paid to both business requirements, and the technical context that the process should be executed in. When concentrating on the business requirements the business process is modelled to closely resemble the business activities and message exchanges. Adding technical constraints to the design process means that the designed executable process needs to be aligned with existing software services. For instance, limited system functionality in existing systems can affect the design of an executable process. Today's vast amount of legacy systems can even result in a design shift where technical aspects to some extent come into focus of the process design. This is especially evident if an executable process language is used to solve system integration issues. The coordination of message exchanges between systems will then be a central aspect of the process.

In this paper, we discuss differences between processes modelled from the pure business perspective and those modelled with considerations of technical aspects.

When designing a process from the *business* perspective, the focus is to "automate the business", i.e. to solve problems that are expressed in business terms. Thus, the aim of the design is to model and later on to implement business processes that capture the message exchanges, activities and roles that are part of a business. For instance, parallel activities in a process will be utilized to denote concurrent business operations. Another example is that a sequence of activities with specific messages to exchange will be used to steer business behaviour according to business contracts. Such processes have the advantage of being easy to understand for people within the business. Another advantage is that the close mapping between the business and the process makes it possible to easily rearrange the activities and sequences when the business changes. Workflows [3] are an example of systems that closely depict the business process as understood by people engaged in the business.

ICSOC'04, November 15–19, 2004, New York, New York, USA.

When designing processes from the *technical* perspective, the focus is to leverage business support by utilizing existing software services. This requires for solving system integration as well as synchronization problems. Just as with modelling processes from the business perspective, the final goal is to support the message exchanges and activities of the business. However, this common goal is reached by integration of existing services. Thus, when constructing technical processes the designer must deal with both business behaviour and the behaviour of existing services. The modelled process will at least partly describe the behaviour of the existing system. For example, it might be decided that parallel activities must be used because the same information is required to be sent to two back-end services. Another example is that a set of alternative activities will be used to enable choice of a system used for communicating a message. These concepts are considerable more "technically" oriented compared to those dealt with when designing pure business processes. Middleware solutions that utilise integration patterns [4] such as message routing, splitting and joining to interconnect back-end systems] are an example of where process descriptions partly deal with system issues rather that business issues. By designing processes from the technical perspective it is possible to coordinate systems interactions in a straightforward way.

In order to build service-based systems that cater to both business needs and technical constraints, it is important to be aware of how a business process, when realized, is affected by existing services. It is also crucial to apply architectures that adhere to the business needs, and at the same time solves technical issues.

In this paper we utilize a process description framework to highlight the differences of processes modelled from the business and technical perspectives. By using the framework, we identify a set of *realization types* that describes possible transformations from a business process into a more technically oriented process. Furthermore, based on the realization types and a notion of lossless realization, we examine under which conditions business and technically oriented processes can coexist in a layered architecture.

The problem of non-fitness between enterprise business processes and information system functionality has been extensively studied in the research community [5, 6, 7, 8, 9]. Our work differs in two aspects. First, we discuss misfit between business and technical perspectives in process modelling, and with focus on executable model specifications. Second, we concentrate our research towards bridging the gap between existing business- and technology-dependent model layers, by introducing a set of transformation patterns. Our usage and comparison of two model layers is similar to the model layers and model transformations in the Object Management Groups (OMG) Model-Driven Architecture (MDA) approach. MDA is a general approach for separating models from the underlying platform technologies [10]. Although the ideas of MDA can be applied to create automated model transformations, for example for component systems [11], MDA is rather a framework for such transformations. Thus, the realization types/transformations that we present in this paper can be used within the MDA framework.

This paper is structured as follows. In the next section we define business and technical processes. Furthermore, we discuss the notion of a technical process as a realization of a business process. In Section 3 an example of a business process is presented, as well as how it is realized following a set of system constraints. In Section 4, we describe a framework that can be utilised to analyze the design and constituents of executable processes. This framework is applied in Section 5 to analyze the possible realization transformations that can be utilized to construct a technical process. Section 6 examines the requirements that need to be followed if a business process and a technical process should co-exist in the same architecture. The paper ends with a discussion of the presented results.

2. BUSINESS AND TECHNICAL PROCESSES

Analysis of a business might result in the design of one or more executable processes. These designs might later be implemented by using an executable process language, such as BPEL4WS [12] or YAWL [13]. Based on the discussion presented in Section 1, we distinguish between two types of process designs:

Business processes (B) are designed based on business concepts such as business events, activities and business actors. This means that the process reflects the pure business perspective.

Technical processes (T) are software realizations of business processes that are designed upon the business concepts and in accordance to existing systems and services. This means the technical process reflects both business perspective and technology concepts such as software services, applications and middleware products.

Given the definitions above, it can be said that technical processes realize business processes by using existing systems. Thus we have the relation that T belongs to the set R(S,B), where R is the function of realization, S are existing system and technology limitations, and T and B are processes that fulfil the same business goals. Since the function R(S,B) can result in several technical processes for the same input (S,B), we do not use the relation T=R(S,B).

If there are no constraints put on the realization from existing systems, the technical process will directly correspond to the business process (T=B). However, the realization can be affected by the following system limitations (S):

1. The business process cannot be implemented as-is, because of domain independent technology limitations, for example limitations in programming languages and communication protocols.
2. The existing domain specific software services, when composed in a process, do not match the documented business process.

Since a business process (as defined above) is defined in business terms, it has the advantage of being easy to understand and modify for people within the business. However, technical processes cannot be overlooked since limitations in existing systems are inevitable.

In the next section we give a concrete example of how system constraints can affect the realization of a business process.

3. EXAMPLE CASE

An example of a business process and its realization as a technical process is shown in Figure 1. The example, in the form of a technical process, is based on a case provided by Sandvik, a global industrial materials engineering company with offices in 130 countries. A major concern of Sandvik is integration and coordination of their existing ERP systems in the form of software services. For this coordination Sandvik utilizes Web service protocols, as well as middleware technology such as Microsoft Biztalk and IBM MQ. As one of the pioneers in the use of Web services, Sandvik has recognized the need to use executable business processes to handle the coordination of its services.

The business process in Figure 1(a) depicts a basic order process where the process is triggered by an incoming order request. Since we are using the Business Process Modelling Notation (BPMN) [14] we depict the customer with a swimlane/pool symbol, message events with encircled envelopes and message flows with dotted arrows. After an order confirmation is sent to the customer, the process is forked into two parallel flows. A shipment plan is constructed while the order is being processed. Later on, the flow is synchronized using an AND-join (called "AND gateway" in BPMN). Before the end of the process a notification that the product has been shipped is sent to the customer.

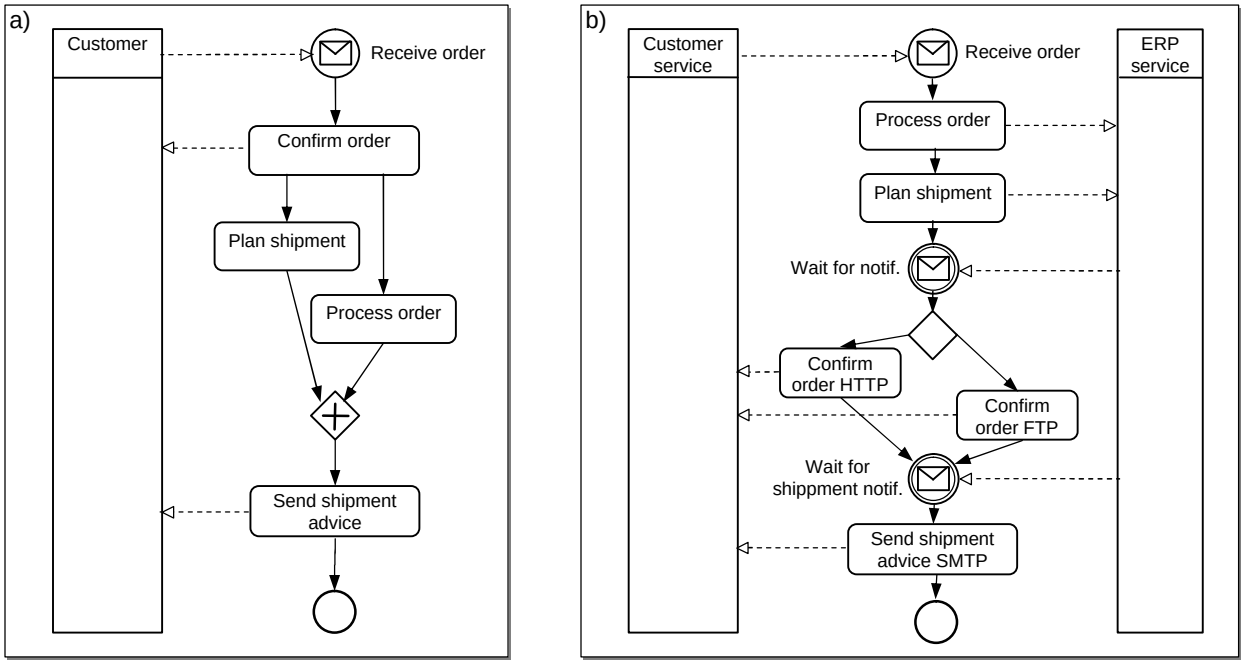

Figure 1. Business process (a) and realized technical process (b).

Figure 1(b) depicts the realized technical process. This process is an excerpt of the process supplied by Sandvik. The technical process is based on existing services, in this case the ERP system (depicted by the ERP service pool in Figure 1) and a service interface to the customers' information systems (the customer service pool in Figure 1). Compared to the business process, the technical process must adhere to a set of system constraints, S:

- S1 – The existing ERP service perform logistics planning and order processing in an integrated activity, a notification can be received when this process is completed.
- S2 – The validation of order information is integrated into the ERP service, order confirmation can be sent when the order is received by the ERP service.
- S3 – Based on the customers (software) service ability order confirmation should be sent as a HTTP message or a FTP file.

These system constraints affect the realization of the business process. The characteristics of the ERP system (S1) prompt us to put shipment planning and order processing in a sequence. Since the message event from the ERP system signals when the order confirmation can be sent (S2), the sending of the order confirmation is placed directly after (instead of before) shipment planning and order processing. Furthermore, the usage of different protocols for sending the order confirmation (S3) prompts us to use an XOR-split (called XOR gateway in BPMN) of the process flow.

Even though the example is small, it shows how a realization of a business process can be affected by system constraints. In this example we used a subset of the process supplied by Sandvik, for brevity reasons we excluded customer authentication and the handling of invoices. We also simplified the handling of protocols, the original process handles more protocols than FTP and HTTP for all activities that notify the customer.

The example elucidates the point that realization does change the design of a process. In the next section we define process design aspects to provide methodical examination of how different types of realizations affect the design of technical processes.

4. ASPECTS OF PROCESS DESIGN

In this section we introduce a conceptual framework to classify different aspects that constitute process design (specification). The framework is based on modeling aspects of workflows, as proposed in [15, 16].

The basic aspect of process specification is *functional*. The functional perspective describes how a process is decomposed, i.e. what activities are to be executed. Functionality of an activity is depicted by name, which uniquely identifies the goal of the activity; by message exchange, designated with input and/or output documents of the activity; and by constraints, which depict activity-internal constraints, such as pre-conditions and post-conditions. The functional aspect, thus, depicts only the semantics of activities. Implementation of activities, however, is not part of the process specification.

The *Behavioral* aspect depicts process control flow, i.e. when an activity is to be executed in relation to others. For specification of dependencies and coordination rules among activities, process specifications rely on a set of basic control flow constructs: sequence, parallel execution (AND split), synchronization (AND join) and conditional branching (OR/XOR split/join).

The *informational* aspect concerns process data. In a process specification, data are information concepts that are used as process attributes upon which flow rules are set and controlled, as well as information that the process exchanges with the external environment. An information concept is depicted by content and structure of contained data.

The *organizational* aspect depicts the distribution of responsibility of executing the activities. In order to model business of an organization, personal and technical resources have to be mapped to specific roles. Roles are related to activities they are intended (allowed) to execute. There may be different relations between roles and activities. For instance, it may be one or more roles allowed to perform an activity; and one role may be allowed to execute one or more activities in a process. By using roles it is possible to dedicate and control responsibilities of parties engaged in a process.

The *transactional* aspect concerns consistent execution and recovery of a set of activities. The design of the transactional aspect includes defining what to be considered as consistent states of the process, thus depicting particular transactional boundaries (scopes). In addition, process transactions may comply to different models, as they contain loosely coupled activities that may have short or long duration. The atomic transaction model [17] is used to administer a set of shorter activities, that all agree to enforce a common outcome by two-phase commit. In contrast, business (long-running) transaction model [18] rules more durable activities, where each activity enforces a globally visible outcome independently of the others; when an activity fails, the parent transaction rolls back to a consistent process state by compensating activities that had successfully completed.

In addition to the defined aspects, by the model of Jablonski and Rausch-Scott, workflows have to cope with other aspects, such as causal and historical (data logging). Those concepts are not, however, used in process specifications; they are related to the workflow environment and monitoring, which we do not consider.

In the next section we describe how each of the above process aspects may be affected when realizing a business process as a technical process.

5. PROCESS DESIGN AND REALIZATION

When designing a business as well as a technical process, each of the five process aspects must be considered. The input to the design is high-level business goals, as well as concrete technical and business requirements and constrains. As mentioned earlier, the input to the design of business processes and technical processes differ, and thus the end result differ.

In this section the primary business and technical design considerations for each of the process aspects is described. Furthermore, a set of *realization types* is identified for each aspect. Each realization type describes a possible transformation from a business to a technical process.

The total set of realization types covers the possible changes that need to be applied to a business process during realization. The selection of the realization types are based on the five process aspects as well as on differences in business and technical process design. The five aspects are used as a fundament to ensure coverage of the properties of processes, while the aspect definitions and the differences in business and technical process design form the basis for selecting realization types on a per-aspect basis.

5.1 Functional Aspect

When comparing the design of business and technical process from the functional perspective we compare how the functionality of the process is decomposed into activities.

Executable business processes cater directly to business needs describing the activities organizations should perform to achieve a common goal. Activities in a business process are modelled, therefore, according to activities performed in organisations and by humans.

When designing technical processes the design issues will shift towards expressing how systems and technology can be combined to solve business problems. In this case the activities are selected according to system operations. Thus, the functionality of existing systems will be the base for selecting the process activities.

Following the definition of the functional aspects in Section 4, activities in a business process could differ from those in the corresponding technical process, with respect to goals, message exchange, and/or imposed pre- and post-conditions. We argue, thus, that the functional aspect of a business process, may be designed in a corresponding technical process by the following realization types:

- Aggregation - an activity *a* in the business process, corresponds to more than one activity (a_1, a_2, a_3,.., a_n) in the technical process, where those activities jointly exchange the same messages as the activity *a*, thus achieving the same goal. As an example, the activity “process order” in the business process may correspond to activities “process order

header by system x" and "process order details by system y" in the technical process.

- Specialization - an activity *a* in the business process, corresponds to more than one activity (a_1, a_2, a_3,.., a_n) in the technical process, where each of those exchange the same messages as the activity *a*, but with different goals due to specialization to a particular system. For instance, as depicted in the example case in Section 3, "confirm order" activity in the business process reflects to the activities "confirm order by HTTP" and "confirm order by SMTP" in the technical process.
- Condition alteration – an activity a in the business process having similar goal and message exchange as an activity a_1 in the technical process, differs from activity a_1 in pre- and/or post-conditions. An example of a difference in pre-conditions that may occur is if an activity in the business process is defined to handle the currencies Euro and Dollar as valid input, while the corresponding activity in the technical process only supports Euro. In this case, the pre-condition of the technical activity is stronger than the corresponding business activity.

5.2 Behavioural Aspect

Comparing the design of the behavioural aspects of technical and business processes involves examining the criteria that governs flow order of activities.

In business processes, the rules that govern the flow coordination of the activities can be stated by simple business rules, such that payment should be done before a product is shipped. Thus, the design of the flow of activities must follow business contracts and other (implied or explicit) agreements of the business partners.

Technical processes must also adhere to the desired business behaviour. The order in which to perform activities might also be governed by limitations in the technical features of the back-end systems, for example by dependencies between existing services. A process that is designed in this fashion might have a quite different ordering of the activities, compared to a process designed from the business perspective only.

Based of the definition of the behavioural aspect in Section 4, and the above discussion, we identify the following realization types that can be applied to transform the behavioural aspect of a business process into a technical process:

- Reordering (concurrency) – sequentially ordered activities in one process (business or technical) may correspond to parallel ordered activities in the other process. For instance, limited transaction capabilities might result in a technical process where an activity that does not support transactions is placed last in a sequence of activities. This design can be placed in contrast to a business process, where the same activities might run in parallel.
- Reordering (sequencing) – activities ordered in a sequence in the one process may be, in the other process, sequenced differentially. For example, contract allows the payment to be done after delivery, but from the technological view, it cannot, as delivery system requires the payment number.
- Condition change - based on a conditional statement, the flow of a process might take different routes. Such a condition might have the same, fewer or more possible branches in a technical process compared to a business process. An example is that a technical process might introduce additional branches to distinguish protocol related issues, such as those depicted in Figure 1(b), where the order may be confirmed either by HTTP or by FTP.

5.3 Informational Aspect

Comparing the informational aspect of business and technical processes can be done by examining the factors that affect concepts of process information as well as their content and structure.

The design of business processes focus on the information need of the business parties, and the need to exchange information between business activities. The concepts in the process information will closely resemble the concepts used in the business (such as "customer", "offer" etc.), and also their content and structure.

The information content used when designing a technical process is the same as those used in a business process, since the process should support required information content in order to support the business. A difference between the business and technical processes is however the structure of the information. The structure simply needs to be adjusted to fit to the existing services and technologies. In addition to the change of structure, the protocols used might require the addition of extra service or protocol specific concepts, such as system or transaction identifiers.

Given the above discussion, we identify three different types of realization of the informational aspect:

- Extension/exclusion – it is possible that the technical process extends or the concepts identified in the business process, for example for handling technology dependent information such as transaction identifiers. Lack of systems supports for some business concepts will also entail the exclusion of concepts in the technical process.
- Projection – concepts defined in the business process might correspond to one or more concepts in the technical process. For instance the concept "product" in the business process may correspond to the concept "item" and "article" in the technical process as those concepts are used by underlying services.
- Redundancy – an information concept in the business process might be used more than once in the technical process. For example, on the technical level a single order might need to be duplicated in order to send it to both to the production and logistic systems.

5.4 Organizational Aspect

Business processes and technical processes utilises the concept of "roles" differently.

An early step when designing a business process is to define the roles that each business party has. Rights to processes and activities are later assigned to the roles. This means that the role concept in a business process is designed according to the business parties that are participating in the process.

In contrast to business processes, technical processes deal with resources and "parties" in form of services provided by systems.

Thus, when guiding activity invocation control, roles are assigned to system services.

Based on the outlined, the organizational aspect of a business process may be transformed in a corresponding technical process by the following realization type:

- Role Mapping – a role in the business process corresponds to several roles in the technical process, or several roles from the business process correspond to a single technical process role. For instance, the business process depicted in Figure 1, a single role, "order facilitator" might be responsible for both "confirm order" and "process order" activities. Invocations of those activities in the technical process might be assigned to different roles – "accepting order system" and "processing order system".

5.5 Transactional Aspect

Comparing two processes from the transactional perspective involves examining similarity in transactional behaviour reflected by specified transaction scopes and models.

The focus when designing a business process is to keep the process aligned with business contract rules. Thus, transaction design is based solely on business criteria. When an error occurs the process must return to a valid state by withdrawing results of completed activities. If by the contract, intermediate results are required to be visible, the process recovery is specified by compensations; otherwise, a valid state is obtained simply by cancelling what is done.

When designing technical processes it must be ensured that no systems are put in an inconsistent state with respect to the overall process. In case of errors the focus will be on service recovery, rather than on business contracts. The recovery methods (atomic vs. compensation) are specified according to business criteria, but also based on characteristics of services (data- vs. non-data based) as well as transaction support of back-end systems.

Thus, transactional aspect of a business process, with existence of supported systems and services, may be transformed in a technical process, by two realization types:

- Scope resize - the boundaries of a transaction differ in the business process and technical process. For instance, following a contract, in the business process it may be required to perform "customer registration", "order acquisition" and "order processing" activities within a transaction, while in the technical process, "customer registration" is not designed as part of the transaction, because underlying service is not transactional (as customer data need not to be deleted if the order is cancelled).
- Model alteration - the model (atomic vs. business) of a transaction differs in the business process and the technical process. As an example, in the business process, it may be required for a transaction, processing an order in several steps (i.e. activities), to provide intermediate results, and accordingly, to roll-back by compensations. However, in the technical process the transaction might be using a two-phase commit mechanism, as the order processing operations are data-based and compensating activities are not, therefore, designed.

As the realization types are defined, some dependencies among them may be observed. For instance, redundancy of an information concept in a technical process (such as processing of a same order by several systems) may be seen as the specialization in the functional aspect. As another example, conditional branching in a technical process may also origin from the specialization to available protocols/services. These examples illustrate that the realization types are related, i.e. that differences in the realization of one aspect may cause changes in another aspect.

Based on the identified differences in realizations of business processes and technical processes, in the following section, we discuss abilities for integration of those two specifications in a software system.

6. LIMITATIONS OF THE LAYERED ARCHITECTURE

As stated earlier, both business and technical processes have advantages. A realization that closely follows the business process will enable easy feedback to business actors in the form of the current process state. By implementing a technical process technical issues such as synchronization can be handled. Ideal would be to have an architecture with two realizations; one "pure" realization of the business processes unaffected by system issues, and one realization of the technical process that handles system issues. An architectural approach that enables this is layering [19].

In a layered approach, the business process is layered on top of the technical process. The executable business process reflects the states in the business, while technical issues are dealt with in the technical process. The dependency between the layers is this case is a "use" dependency [20], thus the executable business process uses the technical process to perform system actions.

The use of layers of executable processes is not new, it has been proposed before as a mean to provide graphical process interfaces to end-users [21]. Yang et al. [22] also describes an approach using "vertical" layers of processes, each layer pertaining to a certain business domain. However, in this chapter we will examine the special case where a layered approach can be applied to join a business and a technical process, without the use of vertical layers.

When executing the layers of processes, the business layer must be synchronized with the technical layer (see Figure 2). A basic criterion for this synchronization is that the technical process must be designed in such a way that it is a *lossless* realization of the business process. A lossless realization of a business process enables the same tracking of business goals as the original process. This means that it must be possible to perform tracking on all five aspects of the business process, even though the technical process is implemented taking system issues into consideration. For example, it must be possible to explain the current (business) process state to a business person even though it is realized as a technical process. As described earlier, the design of the layers adheres to different design principles, thus it might not be possible to use a layered approach in all cases. In the following sections we examine which types of business process realizations that hinder a lossless realization, and thus hinder the use of the layered approach. The examination is based on the previously described realization types.

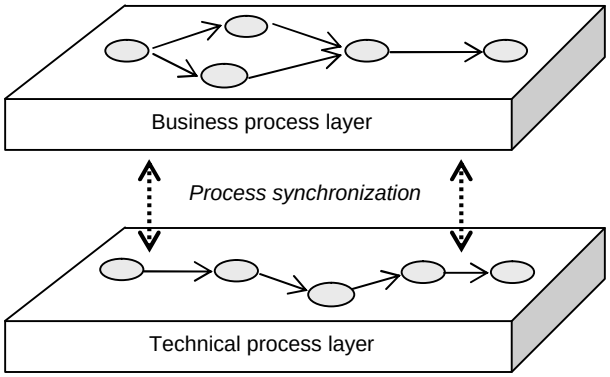

Figure 2. Overview of the layered approach.

Realization of the functional aspect of a business processes can affect its set of activities and their pre and post conditions on the business level. When using a layered architecture, the realization of the business process must adhere to the following design requirements:

- Named activities in the technical process must be *aggregated*/mapped into a single activity name in the business process, otherwise it is not possible to determine which activity that are executing on the business level. Thus, two named activities in the business process cannot correspond to a single activity in the technical process.
- *Conditions* in the technical process must be designed such that the activity pre-conditions are the same or weaker in the technical process. Post-conditions in the technical process must be the same or stronger. Note that both set of conditions will be checked at runtime, since both the business process and the technical process will be executed, each in their own layer. Note also that this design requirement is the same as when designing inheritance hierarchies in object-oriented systems [23].

Note that s*pecialization* of activities in the business process can occur without affecting the synchronization of the processes.

Realization of the behavioural aspect, including sequencing, parallelism and conditional statements can also affect the possibility to use the layered design. With regard to the behavioural aspect the realization must adhere to the following:

- The *concurrency* in form of parallel flows and synchronizations must be designed such that parallel flows are avoided in the technical process, where sequential flow is used in the business process. Without following this requirement, the state of the control flow on the business level cannot be determined.
- The *sequencing* of the activities in the business processes must be reflected in the ordering of the corresponding activities in the technical process. In essence this ensures that the control flow of the activities in the technical process can be depicted by the control flow of the business process.
- *Conditional* control flow must be designed such that each (optional) path in the business process corresponds to at least one unique path in the technical process. If this is not the case it is not possible to determine which path in the business process that is being executed.

The informational aspects concerns realization of the technical process in the form of projection and extension of the concepts present the business process. In addition to this, redundancy on the concept instance level might be introduced. Keeping the technical and business process synchronized with regard to the informational aspect entails following the following design requirements:

- *Exclusion* of concepts in the technical process may not be introduced. However, extensions to handle technical issues might be introduced.
- *Projection* of concepts in the business process must be done such that each concept is represented in the technical process. If the layers contain differences in data naming, a mapping is necessary.

Note that *redundancy* may be introduced in the form of handling duplicated information at the technical level, without affecting the synchronization of the processes.

When realizing the organizational aspects, roles in the business process must be mapped to systems in the technical process. To keep a lossless realization this mapping must adhere to the following:

- Parties in the business process that is responsible for executing the business activities must own, or be responsible for, the realizations of these activities in the technical process. For example if a party is responsible for an activity on the business level, and that activity is performed by a system owned by another party, it is unclear who got the actual responsibility for the correct outcome of the activity.

The transactional aspect concerns the realization of transaction models and the design of transactions boundaries. In order to provide the same transactional integrity as the business process the realization must:

- The *transaction model* of the activities in the in the technical process must support the same, or a higher level of transaction model compared to the business process. For example, if the business process is designed using long running transactions, the technical process must support atomic transactions or long running transactions.
- The *transaction boundaries* must be realized such that the boundaries in the technical process do not overlap those defined in the business process.

As presented above, restrictions on the use of most realization types must be introduced to yield a lossless realization. To achieve a lossless realization of all aspects of a business process is therefore difficult. For example, the real-world process in Figure 1 is not a lossless realization. In this case, a comparison of the behavioural aspect of the business and technical process reveals a use of the sequencing realization type that is not lossless. Even in those cases a total “losslessnes” can not be achieved, awareness of the realization types can lead to a process design that is closer to the original business process.

7. CONCLUSION

In this paper we have addressed the realization gap between business and technical processes. On an abstract level this gap can be described as a business versus information technology gap. However, our contribution lies on a more concrete level as we, based on a process description framework, define a set of realization types that describe the possible transformation needed when constructing a technical process from a business process. In addition, we have presented design requirements that need to be followed in order to create a lossless realization of a business process.

This work has been conducted in several steps. It started with the process case presented in Figure 1. When examining this process, it was evident that the expressiveness of the process notation was used to cover both business and technical needs. This motivated us to make a structured examination of how technical considerations affect realization of a business process. The second step of the work was to examine under which conditions a business process and a technical process could be integrated into a single architecture. The first part of the work resulted in the definition of business and technical processes presented in Section 2, as well as the realization types presented in Section 5. The second part of the work resulted in the notion of lossless realization, and the design requirements for lossless realization presented in Section 6.

The presented result can be applied in several ways. Firstly, the definition of business and technical process along with the presented realization types can be used as a conceptual framework for discussing executable process design in the context of existing systems. Secondly, the presented guidelines on lossless realization can be used to govern the design of software architectures that enable a closer integration between the business and technological view of processes.

The presented realization types are a first step in creating a model-driven tool that supports realization of business processes. Further work with the aim to create such a tool entails examining the relations between the realization types, as well as a closer examination of the parameters of the realization function, R(S,B). Additional examination of the realizations types includes their formalization, and describing dependencies among them. Some dependencies among realization types are briefly mentioned in the end of Section 5. However, these dependencies and the order in which to apply realization types need to be further examined.

Considering future work with the realization function (R), we defined it as having two parameters, the business process and system constraints, R(S,B). However, in this paper we have not examined the relative importance of system constraints (S) and the business process (B). Neither have we categorized the system constraints. As mentioned in the introduction, there might be cases where the power of executable process languages is used to solve mainly technical issues. In these cases technical issues (S) in the realization process might actually have a bigger influence on the final outcome than business aspects (B). Future categorization of system constraints might enable relating the realization types with typical system constraints, thereby forming the basis for tool based business-to-technical process transformations.

8. ACKNOWLEDGMENTS

The authors would like to thank Sandvik for providing the input to the example case presented in Section 2. This work is a part of the Serviam project, partly funded by the Swedish Agency for Innovation Systems.

9. REFERENCES

[1] Papazoglou, M., Georgopoulos D. Special issue on service oriented computing. Communications of the ACM, 10, 46 (Jan. 2003), 24-28.

[2] Piccinelli G., Zirpins, C., Lamersdorf W. The FRESCO Framework: An Overview. Proceedings of the 2003 Symposium on Applications and the Internet Workshops. IEEE Computer Society (2003), 120-126.

[3] Sharp, A., McDermott, P. Workflow Modeling. Artech House, Inc., Boston, USA, 2001.

[4] Hohpe, G. et al. Enterprise Integration Patterns, Addison. Wesley, Oct 2003.

[5] Bubenko, J.A., Jr., Wangler, B. Objective driven capture of business rules and of information systems requirements. In Proceedings of the IEEE Systems Man and Cybernetics '93 Conference (Le Touquet, France, October 1993), 670-677.

[6] Grover, V., Fiedler, K., Teng, J.T.C. IEEE Transactions on Engineering Management, 41, 3 (Aug. 1994), 276 – 284.

[7] Yu, E. S. K., Du Bois, P., Dubois, E., Mylopoulos, J. From Organization Models to System Requirements: A 'Cooperating Agents' Approach. In Proceedings of the Third International Conference on Cooperative Information Systems (CoopIS'95) (Vienna, Austria, May 1995), 194-204.

[8] Rolland, C.,Prakash, N. Bridging the Gap Between Organisational Needs and ERP Functionality. Journal of Requirements Enigneeering, 5, 3, 2000, 180-193.

[9] Rolland, C.,Prakash, N. Matching ERP System Functionality to Customer Requirements. 5th IEEE International Symposium on Requirements Engineering (RE 2001) (Toronto, Canada, Aug. 2001). IEEE Computer Society (2001), 66-75.

[10] Kleppe, A., Warmer, J., Bast, W., MDA Explained, Addison-Wesley, Apr 2003.

[11] Weis, T., Ulbrich, A., Geihs, K. Model Metamorphosis. IEEE Software (sept./oct. 2003), 46-51.

[12] BEA, IBM, Microsoft, SAP and Siebel. Business Process Execution Language for Web Services (BPEL). http://www-106.ibm.com/developerworks/library/ws-bpel/, June 9 2004.

[13] Aalst van der, W., Hofstede, A., Aldred, L. Yet Another Workflow Language (YAWL). http://www.citi.qut.edu.au/yawl/index.jsp, June 9 2004.

[14] White, S. Business Process Modeling Notation Version 1.0, The Business Management Initiative, May 2004.

[15] Jablonski, S. A Software Architecture for Workflow Management Systems. In Proceedings of the Ninth International Workshop on Database and Expert Systems Applications (DEXA'98) (Vienna, Austria, August 1998). . IEEE Computer Society, 1998, 739-744.

[16] Rausch-Scott, S. TriGSflow – Workflow Management Based on Active Object-Oriented Database Systems and Extended Transaction Mechanisms. PhD Thesis, University at Linz, 1997.

[17] Bernstein, P., Hadzilacos, V., Goodman, N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, 1987.

[18] Garcia-Molina, H. Modeling Long-Running Activities as Nested Sagas. IEEE Data Engineering Bulletin, 14, 1, 1991, 14–18.

[19] Bass, L., Clements, P. and Kazman, P. Software architecture in practice. Addison Wesley, 1998.

[20] Parnas, D. Designing software for ease of extension and contraction. IEEE Transactions on Software Engineering, March 1979, 128-138.

[21] Johannesson P., and Perjons, E. Design principles for process modelling in enterprise application integration, Information Systems, 26, 2001, 165-184.

[22] Yang, J., Papazoglou, M. Interoperation Support for Electronic Commerce. Communications of the ACM 6, 43, 2000, 39-47.

[23] Meyer, B. Applying Design by Contract. IEEE Computer, 25, 10, (Oct. 1992), 40 – 51.

Paper C

Architectures for Service-oriented Processes

Martin Henkel and Jelena Zdravkovic

The Nordic Conference on Web Services (NCWS'04), Växsjö, Sweden, November 22-23, 2004.

Architectures for Service-oriented Processes

Martin Henkel[1], Jelena Zdravkovic[2]

[1] Stockholm University and Royal Institute of Technology,
Department of Computer and System Sciences, Forum 100,
164 40 Kista, Sweden
martinh@dsv.su.se
http://www.dsv.su.se

[2] University of Gävle and Royal Institute of Technology,
Department of Computer and System Sciences
801 76 Gävle, Sweden
jzc@dsv.su.se
http://www.hig.se

Abstract. By the use of Web Service technologies and the Internet it is possible to lay the foundation for virtual value chains that cross enterprise boundaries. As the number of services and their interaction grow, it is evident that the flow of message exchange between services needs to be coordinated in a structured way. Executable process languages such as BPEL are proposed as an instrument for the coordination of services. Executable processes must be designed such that they solve technical coordination problems as well as provide a fundament for organizations to manage and monitor the progress of the business. In this paper we examine how the design of executable processes is affected by both technical and business issues. Furthermore, we examine a set of architectures that enable the use of executable processes to cater to both business and technical needs. We provide fundamental guidelines on how to apply the architectures.

1 Introduction

Software services are the building block of future systems that integrate existing IT assets and thereby provide the basis for building complex cross-enterprise systems. Even though the Web Service technologies SOAP and WSDL solve many interoperability issues, they do not provide support for the complex task of coordinating the execution of software services. Coordination of services involves executing and monitoring complex message exchanges, as well as to provide massive parallelism and robust error handling. The core of the coordination problem lies in controlling the dynamic aspect of service execution. Thus, when examining the coordination problem, the focus is on dynamic aspects such as the order of message exchange and the timing of messages, rather than static aspects of services such as interfaces. Since process descriptions focus on the dynamic aspects, it is natural to use processes as a tool for describing and solving coordination problems. Future systems

will increasingly rely on composing and coordinating individual services to create new, complex business interactions in the form of processes [1]. Executable process languages such as the Business Process Execution Language for Web Services (BPEL4WS,[2]) are specifically targeted towards solving coordination problems.

A successful process implementation relies on that the executable process can capture business aspects as well as handle technical details. Processes modelled to close resemble a business will depict the activities, message exchanges and rules of an organisation. When this *business process* is to be realised in a system, additional technical issues such as system interoperability and message synchronization must be solved. Thus, we identify two perspectives of process design, the business perspective and the technical perspective. Seen from the business perspective, executable processes offer the ability to monitor and alter the processes according to business rules. Workflow systems [3] are an example of processes used from the business perspective. Taking the technical perspective, executable processes offer the ability to perform complex system integration and synchronization. An example of the technical perspective is the numerous messaging patterns [4] that exist to handle system coordination.

In order to cater to both business and technical needs, the business and technical perspectives must coexist in future system architectures.

In this paper we examine the properties of processes designed from the technical and business perspectives. We base this examination on a conceptual framework for process specifications. Furthermore, we examine three architectures that can help merge the two perspectives of processes. We also provide basic guidelines on how to apply the architectures. The aim of the paper is to provide an overview of different process perspectives, and provide basic guidelines on how to merge them. The work presented in this paper is a continuation of previous work presented in [5].

This work specifically focuses on technical and business perspectives of executable processes. On a much more general level, this problem has been studied by other authors as a miss-fit between business and information system functionality ([6], [7], [8]). Our use of two perspectives on processes also bears some resemblance with the model layers proposed by the Object Management Groups (OMG) Model-Driven Architecture (MDA) approach. However, MDA uses model layers to separate models from platform technologies [9]. Compared to the work presented in this paper, MDA is thus to be considered as a general framework for model transformations.

This paper is structured as follows. In the following section we define and give a concrete example of business and technical processes. In Section 3, we discuss design differences between the two process types. Section 4 introduces three architectures that combine technical and business issues. Guidelines on how to apply the architectures and conclusions are presented in section 5 and 6.

2 Business and Technical Processes

Analysis of a business might result in the design of one ore more executable processes. These designs might later be implemented by using an executable process language, such as BPEL4WS. Based on the discussion presented in the Introduction section, we distinguish between two types of executable process designs:

Business processes directly corresponds to the processes that where documented during analysis. This means that the executable process reflects the pure business perspective, containing the same activities, roles etc as the business process.

Technical processes are designs that do not correspond directly to the documented business processes. This kind of processes is designed to adhere to business as well as technical requirements. The motivation for creating a technical process that does not resemble the business process is:

1. The business process cannot be implemented as-is, because of technology limitations, for example in programming languages and communication protocols.
2. The existing software services, when composed in a process, do not match the documented business process.

Since an executable business process (as defined above) is defined in business terms, it has the advantage of being easy to understand and modify for people within the business. However, technical processes cannot be overlooked since limitations in existing system are inevitable.

An example of a business process and its realization as a technical process is shown in Figure 1 (this example is also presented in [5]). The example, in the form of a technical process, is based on a case provided by Sandvik, a global industrial materials engineering company with offices in 130 countries. A major concern of Sandvik is integration and coordination of their existing ERP systems in the form of software services. As one of the pioneers in the use of Web services, Sandvik has recognized the need to use executable business processes to handle the coordination of its services.

The business process in Figure 1(a) depicts a basic order process where the process is triggered by an incoming order request. Since we are using the Business Process Modelling Notation (BPMN) [10] we depict the customer with a swimlane/pool symbol, message events with encircled envelopes and message flows with dotted arrows. After an order confirmation is sent to the customer, the process is forked into two parallel flows. A shipment plan is constructed while the order is being processed. Later on, the flow is synchronized using an AND-join. Before the end of the process a notification that the product has been shipped is sent to the customer.

Figure 1(b) depicts the realized technical process. This process is an excerpt of the process supplied by Sandvik. The technical process is based on existing services, in this case the ERP system (depicted by the ERP service pool in Figure 1) and a service interface to the customers' information systems (the customer service pool in Figure 1).

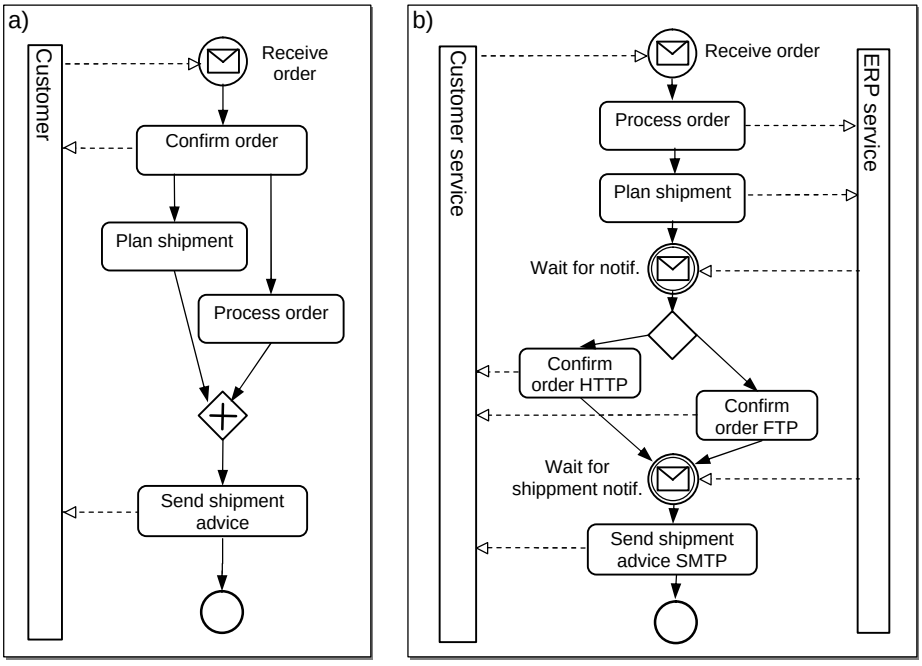

Fig. 1. Example business process (a) and realized technical process (b)

Compared to the business process, the technical process must adhere to a set of system constraints:

- S1 – The existing ERP service perform logistics planning and order processing in an integrated activity, a notification can be received when this process is completed.
- S2 – The validation of order information is integrated into the ERP service, order confirmation can be sent when the order is received by the ERP service.
- S3 – Based on the customers (software) service ability order confirmation should be sent as a HTTP message or a FTP file.

These system constraints affect the realization of the business process. The characteristics of the ERP system (S1) prompt us to put shipment planning and order processing in a sequence. Since the message event from the ERP system signals when the order confirmation can be sent (S2), the sending of the order confirmation is placed directly after (instead of before) shipment planning and order processing. Furthermore, the usage of different protocols for sending the order confirmation (S3) prompts us to use an XOR-split of the process flow.

Even though the example is small, it shows how a realization of a business process can be affected by system constraints. In this example we used a subset of the process supplied by Sandvik, for brevity reasons we excluded customer authentication and the handling of invoices. We also simplified the handling of protocols, the original

process handles more protocols than FTP and HTTP for all activities that notify the customer.

The example elucidates the point that realization does change the design of a process. In the next section we define process design aspects to provide methodical examination of how a business process might differ from a technical process.

3 Designing Business and Technical Processes

In this section we examine the design of business and technical processes to identify possible differences in respective process designs. To make a structured examination, we use a conceptual framework that identifies four aspects of process design. The framework is based on modelling aspects of workflows, as proposed in [11] and [12]. The first basic aspect of the framework is the *functional* aspect; it describes how a process is decomposed into activities, i.e. what activities are to be executed. The *behavioural* aspect depicts process control flow, i.e. when an activity is to be executed in relation to others. The *informational* aspect concerns content and structure of process data. The *transactional* aspect concerns consistent execution and recovery of a set of activities.

When designing a business as well as a technical process, each of the process aspects must be considered. In the rest of the section we describe how each of the four aspects may be affected when realizing a business process as a technical process. For a more detailed description of the design differences between technical and business processes we refer to [5].

Functional Aspect When comparing business and technical processes from the functional perspective we compare how the functionality is composed into activities. We also must consider the pre- and post-condition of the activities. Executable business processes cater directly to business needs, activities in a business process will thus be designed according to the activities in the organization. When designing technical processes the design issues will shift towards expressing how technology and existing system operations can be combined to solve business problems. The functionality of existing systems will be the base for selecting the activities for inclusion in a technical process. These design differences might result in the following differences that may appear in the functional aspect of a business and a technical process:

An activity in the business process corresponds to more than one activity in the technical process, where those activities jointly achieve the same goal as the activity in the business process.

Activities in the business and the technical process having similar goal differ in pre- and/or post-conditions.

Behavioural Aspect Comparing the behavioural aspects of technical and business processes involves examining the criteria's that governs the process control flow. In business processes, the rules that govern the flow coordination of the activities are stated by business rules, such that payment should be done before a product is

shipped. Technical processes must also adhere to the desired business behaviour. However, the order in which to perform activities might also be governed by the technical features of the back-end systems, for example by dependencies between existing services. Due to these design differences the processes might differ in sequencing (including parallelism) and in the use of conditional branching. We, therefore, identify the differences that may appear in the behavioural aspect of a business and a technical process as the following:

Sequentially ordered activities in one process (business or technical) may correspond to differently sequenced or parallel ordered activities in the other process.

A condition might have the same, fewer or more branches in a technical process compared to a business process.

Informational Aspect The design of the informational aspect concerns the information structures used by the process. The design of business processes focus on the information need of the business parties; the process information will closely resemble the concepts used in the business. The information content used when designing a technical process is based on the business concepts. A difference between the business and technical processes is however how the information is structured. In the technical process, the structure simply needs to be adjusted to fit to the existing services and technologies. In addition to the change of structure, the protocols used might require the addition of extra service or protocol specific concepts, such as system or transaction identifiers. Based on this, we identify the differences that may appear in the informational aspect of a business and a technical process:

A technical process may extend/exclude the concepts identified in the business process.

Concepts defined in the business process might correspond to one or more concepts in the technical process.

Transactional Aspect Comparing two processes from the transactional perspective involves examining similarity in transactional behaviour reflected by specified transaction models (atomic vs. long-running [13]) and transaction boundaries. When an error occurs the process must return to a valid state by withdrawing results of completed activities (by compensation for long-running transactions or by simple cancelling for atomic transactions). The focus when designing a business process is to keep the process aligned with business contract rules. When designing technical processes it must be ensured that no systems or services are put in an inconsistent state with respect to the overall process Thus, transactional aspect of a business process, with existence of supported systems and services, may differ from a technical process in following:

The boundaries of a transaction may differ in the business process and technical process.

The model (atomic vs. long-running) of a transaction differs in the business process and the technical process.

In Table 2, below, we summarize the design of technical and business processes for the described process aspects.

Table 2. A summary of design aspects for business and technical processes

	Functional	*Behavioural*	*Informational*	*Transactional*
Design task	Decompose into activities	Sequencing of activities	Structure message and process information	Select transaction model and transaction boundaries
Business process design	Decompose according to business activities	Governed by business rules and contracts	According to business concepts	Keep consistent business states
Technical process design	Decompose according to operations in existing services	Governed (partly) by features of the existing systems	Business concepts split/extended according to service needs	Keep systems and data in a consistent state

Based on the identified differences in realizations of business processes and technical processes, in the following section, we discuss abilities for integration of those two perspectives in a software system.

4 Architectures

Software architectures need to support both business and technical perspectives. Ideal would be to have a software architecture that includes both the business and technical perspectives of executable processes. Architectures that combine business and technical perspectives of processes must be able to handle all, or at least some of the process design differences that where discussed in Section 3. An example of a difference is that a technical and a corresponding business process might contain different number of activities (i.e. they differ in the functional aspect). Even though this difference exists, the architecture must be able to convey the state of the business process, for example by making it possible to instantly identify the activities that are currently executing on the business level.

When constructing an architecture that support both business and technical aspects there are basically three possible approaches:

a) Use the designed technical process as a starting point and add the features of the business process. We call this architecture the Layered architecture.

b) Use the business process as a fundament, and add technical aspects. This will result in what we call an Aspect oriented architecture.

c) Combine business and technical aspects by identifying “modules” that encapsulates technical or business behaviour. This architecture is named the Domain service architecture.

In the following sections we will examine how each of the three architectures enables a closer integration between business and technical processes. We base our

examination on the four process aspects (functional, behavioural, informational and transactional) described in section 3.

4.1 The Layered Architecture

In a layered architecture, the business process is layered on top of the technical process (see Figure 2). The executable business process reflects the states in the business, while technical issues are dealt with in the technical process. The two layers of processes is kept synchronized, for example by implementing a publish-subscribe event system.

Generally the business process will represent an abstraction of the underlying technical process. This is in-line with the common architectural principle of abstraction layers [14]. The business process layer is dependent on the technical process for executing systems actions, thus this dependency can be categorised as a "use" dependency [15].

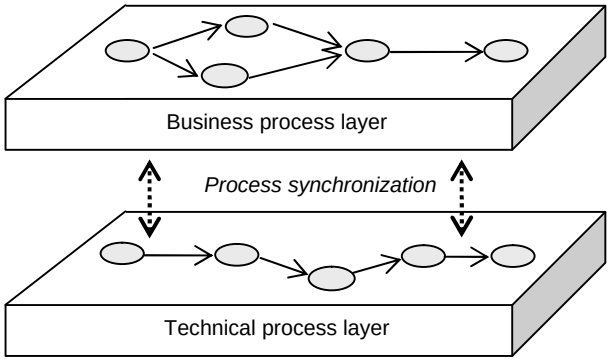

Fig. 2. Overview of the layered architecture

A prerequisite to using this architecture is that it should possible to perform process synchronization between the two layers. In essence the design of all of the four aspects of the technical process must be done such that it is possible to keep the four aspects of the business process unaltered.

The *functional aspect* of a process contains its activities and the pre-and post conditions of those activities. In the technical process layer it is possible to break down a business activity into several technical activities without disrupting the synchronization. Following principles from object-oriented programming [16] pre conditions in the technical process can also be weakened in the technical process, while post-conditions can be made stronger. However, to maintain synchronization between the two layers, the functional aspects must be designed such that an activity in the business process corresponds to at least one (unique) activity in the technical process. For example, if two activities in the business process correspond to a single technical activity, it is not possible to determine which of the business activities that are executing.

The *behavioural aspect* of a process refers to the order of execution of its activities. The layered architecture enables the technical process to contain advanced parallel execution and synchronisation of activities, this enable for example gathering information from several back-end systems. Just as for the functional aspect, the behavioural aspects of the technical process cannot deviate too much from the behaviour of the business process. A basic criteria for proper synchronization of the processes is that the order of the activities in the technical process follows the same order as in the business process. Similarly, parallel flows of activities cannot be employed on the technical level if the corresponding activities on the business level must be executed in sequence. Conditional branches in the business process flow should be represented with at least the same amount of branches in the technical process. Without following these basic criterias, the business process will behave inconsistent.

Synchronizing the *informational aspect* of the two processes mean that business information present at the business layer must be possible to extract from the technical layer. The technical layer can contain additional concepts for handling technical issues, such as transaction identifiers. In the technical layer it is also possible to duplicate the information used in the business process, this might be useable if the information is stored in several systems. A basic requirement of the layered architecture is however that the technical layer must be able to represent all the concepts of the business process.

The *transactional aspects* of the two layers describe the transaction model of the processes, as well as transaction boundaries. By using two layers it is possible to introduce two-phase commit protocols (atomic transactions) on the technical layer, even thought the business level are designed by using long-running transactions. To perform proper transaction synchronization between the layers, the transaction model in the technical layer must support the same, or a higher level of transaction model compared to the business process. For example, if the business process is designed using long running transactions, the technical process must support atomic transactions or long running transactions. Another limitation of this architecture is that the transaction boundaries of the two levels must be designed such that they do not overlap. For example, two business activities grouped together into a transaction should not be executed as two unrelated transactions on the technical level.

The main advantage of this architecture is its clear separation between business and technical issues. This separation makes it possible to represent a graphical overview of the business process, while still enabling detailed monitoring of the technical process. However, as described above, the synchronization of the levels can introduce severe limitations of the process design. Simply put, the business level must be a “pure” abstraction of the technical level. In many cases it is not possible to realize the technical level without affecting the business level. An example of this is the business and technical processes shown in Figure 1. In the example, the two processes differ too much in the behavioural aspect to be able to coexist in a layered architecture (the order of execution of the activities differs).

6.2 The Aspect Oriented Architecture

While the layered architecture has the technical process as the basic building block, the aspect oriented architecture starts with the business process as the fundament. The main idea of this architecture is to add orthogonal technical features to the business process without affecting its design. Commonly these orthogonal technical features can be expressed as non-functional requirements [17] such as security, transaction handling and error handling.

When the business process is executed the functionality of the aspects is combined with the process. This process of at runtime extending code with aspects is commonly called “Weaving” [18].

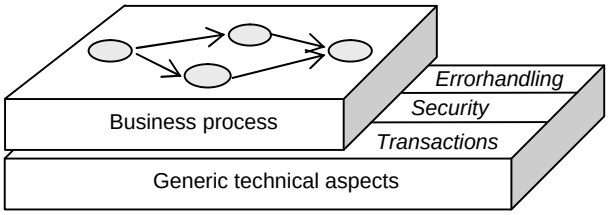

Fig. 3. Overview of the aspect architecture

Aspects such as those presented in Figure 3 are not the only types of functionality that can be expressed as aspects. Any functionality that is generic enough to be independent of the actual design of the business process can be added as an aspect. Just as for the layered architecture, the aspect architecture cannot be applied to implement all business processes. In the following we examine each of the four process aspects to single out possible uses of the aspect oriented architecture, and to find its limitations.

The *functional and behavioural aspects* of a process can partly be realized by using the aspect oriented architecture. As an example, the technical process depicted in Figure 1 contains activities that handle FTP and HTTP communication. The use of these two protocols affect both the functional (number of activities) and the behavioural (the conditional process flow) aspect of the process. By constructing an aspect that handles protocol selection and invocation it is possible to remove both the functional and (some of) the behavioural differences of the processes shown if Figure 1. The constructed aspect would be invoked whenever a message should be sent to the customer. The aspect then selects, dependent on the customer information, the appropriate protocol to use (FTP or HTTP). However, aspects cannot replace the domain specific parts of the technical process. For example, it is not feasible to construct an aspect that handles the difference in the ordering of the activities in the process.

Technical additions in the *informational aspects* can be handled by weaving aspects into the business process. An example of this could be that certain processes need to be identified with a process identifier on the technical level. An aspect can be constructed such that upon instantiation of the process a unique process identifier is

generated. This process identifier can subsequently be used as and identifier when connecting to existing back-end systems. However, adding information structures like this is delimited to generic, domain agnostic information. For example, adding the creation of an order number is not as simple as creating a “technical” identifier. The creation of an order number might be tightly governed by business rules, and thus cannot be handled on a generic level.

The *transactional aspect* concerns transaction model and transaction boundaries. Generic handling of transactions is a typical example of where the aspect oriented approach can play a major role. In fact, technologies that provide automatic transactions handling is already in wide use. Examples are Enterprise Java Bean (EJB) component servers and Microsoft enterprise services (formerly COM+). These technologies effectively remove the burden of handling transactions from the developer. For example automatic transformation from atomic to long-running transaction can be provided by generic aspects. The design of the transaction boundaries, however, still needs to be decided and implemented by the process designer.

The main advantage of this architecture is that technical features can be moved into generic aspects, or middleware products. These aspects and products can then be applied to a wide range of business processes.

Overall the aspect-oriented architecture is a very promising approach to remove domain-independent, generic technical functionality out of the core business process. While there exist special aspect-oriented languages, such as AspectJ [19], the aspect can also be provided by middleware servers. Business Process Management Systems (BPMS) provide basic aspects such as scalability and persistence. In the future use of the Business Process Execution Language for Web Services (BPEL4WS) will enable middleware servers to add more features (like secure communication) to processes.

6.3 The Domain Service Architecture

The two previously described architectures started with either the technical process or the business process as a fundament. A third approach to creating an architecture that support both business and technical perspectives is to combine technical and business requirements at the design stage. A viable solution is to let technical and business requirements guide the creation of modules that later can be combined using an executable process language. This approach is commonly presented as an option when building process-based systems [20], [21], [22]. Figure 4 contains a schematic overview of a domain service architecture where the modules sales, production planning and logistics are coordinated by a process layer.

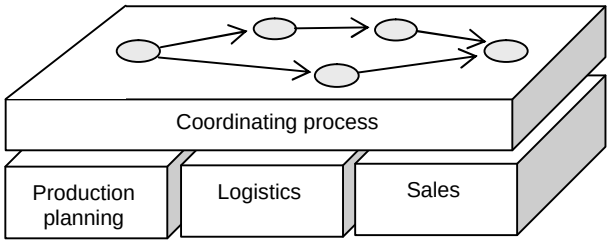

Fig. 4. Domain service architecture

The advantage of this architecture is that it is possible to hide technical details in the modules, and thereby only expose a façade to the modules functionality. The critical point when employing this architecture is to select a principle that governs the decomposition of functionality into modules. Selecting modules according to existing business functions are proposed by some authors [20], [23]. This approach is illustrated in Figure 4. Another option commonly used when construction component-based systems is to partition the modules according to central business concepts [24][25]. For companies with a large set of legacy systems, existing systems might also govern the creation of modules. In the following we will examine possible ways to implement the four aspects of processes using the service domain architecture.

The ability to represent the *functional* and *behavioural* aspects of a business process is highly dependant on the criteria of decomposition. The granularity of the modules and their interfaces decide the level of control the coordinating process can have over the modules. For example, a module might encapsulate several business activities and represent them as a single operation in the module interface. This encapsulation will affect the design of the coordinating process, the level of control will be lover at the process level. The example also holds for the behavioural aspect, where complex activity dependencies can be encapsulated in a module. The key issue here is to design the modules such that they encapsulate technical functions and behaviour, rather than hiding high-level business functions and behaviour.

The *informational* aspect of processes in the service domain architecture concerns the information content that each module exposes to the coordinating process. Using this architecture, it is simple to encapsulate technical information in each module. A risk when using this architecture is that technical information concepts will affect the coordinating process. An example of this is a module that requires a costumer's digital certificate to send a notification. The certificate must be supplied by the process, since communication between modules would break the architecture. This implies moving certificate handling to the process level, which might be undesired.

Handling the *transactional aspects* can be done by firstly implementing module-internal transactions, and secondly by letting each module implements a transactional interface. The transactional interface lets the coordinating process coordinate transactions on the business level. A key issue of this architecture is to handle differences in transactional models between modules. Basically, conversion of transactional protocols can be handled either in the process on in each module.

While the service domain architecture provides a good way to isolate technical details into modules, it relies heavily on the selection of decomposition criteria for the modules. If technical considerations play a major role when designing the modules, it will affect the coordinating process. Since all communication go through the coordinating process there is a risk that technical features “creep” into the process. Note that this problem is not evident in the layered architecture, since that architecture enables communication on the technical level.

7 Selecting Architecture

The three architectures to some extent represent extremes, when selecting architecture the best option is to combine features of the architectures.

The layered architecture can be used to provide monitoring capabilities for the business, while enabling complex technical processes on a separate layer. As discussed before, the needed process synchronization is the Achilles heel of this architecture. In practice it might not be possible to implement his architecture in its pure form. However, if the process synchronization is made less strict, the architecture is applicable in more cases. A “loosened” synchronization can be applied by allowing inconsistencies on the business level process. This would yield an executable process that indicates the status of the process, rather than to directly correspond to the executing process.

The aspect oriented architecture is applicable as a powerful tool for introducing system-wide technical properties. However, domain specific details should not be implemented as aspects, since there is not point in adding the complexity of an aspect to implement functionality that only concerns a single activity.

The service domain architecture provides a traditional fundament to build systems by dividing the functionality into modules. This is of course applicable to all processes. However, the architecture lacks the clear separation of processes provided by the layered architecture. The architecture also lacks the possibility to add generic technical functionality.

Given the above discussion, we propose that the features of all the architectures should be combined in the following order:

1) Maximize the use of aspects by identifying technical concerns that affects a large part of the activities. Use an aspect oriented language, or an existing middleware product to supply these aspects.
2) Construct a domain service architecture by building modules (services) that represent existing business concepts or functions. This enables the coordinating process to be as close to the business process as possible.
3) If needed, supply a business process monitor by implementing a second process layer depicting a simplified business process. To make this possible, the synchronization criteria of the processes must be relaxed.

Applying the features of the architectures in the above order makes it possible to maximize the advantages of the architectures.

8 Conclusion

In this paper we initially identified two types of processes, business and technical. These processes cater to partially different needs, and are thus designed differently. The possible differences in process design where examined using four aspects of process design. Furthermore, we used the identified process differences to examine three architectures that aim to unify business and technical aspects. The three architectures represent three different ways to deal with the integration of technical and business issues in executable processes. While these architectures can be applied directly, we proposed simple guidelines on how to combine the architectures.

Having a clear separation between business and technical issues is important when building an executable process. Our definitions of process types and the examination of their design differences can contribute to a better separation of the process types, and thus aid in the design process. The presented architectures and guidelines can aid this design process further. The architectures are not novel, our contribution is rather to examine them in the context of executable processes.

The three architectures represent three distinct ways to unify technical and business perspectives in a single architecture. The layered architecture and the aspect oriented architecture focused on the business and technical aspects respectively. The domain service architecture is an architecture where technical and business aspects can be "mixed" together. These architectures are on an abstract level, and must thus be combined with other architectural principles to build large scale enterprise systems. In particular, we have not discussed the possibility to "mix" technical and business issues in more than two abstraction layers.

Further work entails examining lover-level architectural principles that can be applied when designing executable processes.

ACKNOWLEDGMENTS

The authors would like to thank Sandvik for providing the input to the example case presented in Section 2. This work is a part of the Serviam project, partly funded by the Swedish Agency for Innovation Systems.

References

1. Piccinelli G., Zirpins, C., Lamersdorf W.: The FRESCO Framework: An Overview. Proceedings of the 2003 Symposium on Applications and the Internet Workshops. IEEE Computer Society (2003), 120-126
2. BEA, IBM, Microsoft, SAP and Siebel. Business Process Execution Language for Web Services (BPEL4WS). http://www-106.ibm.com/developerworks/library/ws-bpel/, (June 9, 2004)
3. Sharp, A., McDermott, P.: Workflow Modeling. Artech House, Inc., Boston, USA (2001)
4. Hohpe, G. et al.: Enterprise Integration Patterns, Addison. Wesley (2003)

5. Henkel, M., Zdravkovic, J., Johannesson, P., "Service-based Processes - Design for Business and Technology", Accepted for the International Conference on Service Oriented Computing, New York (2004)
6. Bubenko, J.A., Jr., Wangler, B.: Objective driven capture of business rules and of information systems requirements. In Proceedings of the IEEE Systems Man and Cybernetics Conference. Le Touquet, France (1993), 670-677
7. Grover, V., Fiedler, K., Teng, J.T.C.: IEEE Transactions on Engineering Management, Vol. 41, Iss. 3 (1994). 276–284
8. Rolland, C.,Prakash, N.: Bridging the Gap Between Organisational Needs and ERP Functionality. Journal of Requirements Enigneeering, Vol 5, Iss. 3 (2000). 180-193
9. Kleppe, A., Warmer, J., Bast, W.: MDA Explained, Addison-Wesley (2003)
10. White, S.: Business Process Modeling Notation Version 1.0, The Business Management Initiative (2004)
11. Jablonski, S.: A Software Architecture for Workflow Management Systems. In Proceedings of the Ninth International Workshop on Database and Expert Systems Applications (DEXA'98). Vienna, Austria. IEEE Computer Society (1998). 739-744.
12. Rausch-Scott, S.: TriGSflow – Workflow Management Based on Active Object-Oriented Database Systems and Extended Transaction Mechanisms. PhD Thesis, University at Linz (1997)
13. Garcia-Molina, H.: Modeling Long-Running Activities as Nested Sagas. IEEE Data Engineering Bulletin, Vol. 14, Iss. 1, (1991). 14–18
14. Bass, L., Clements, P. and Kazman, P.: Software architecture in practice. Addison Wesley (1998)
15. Parnas, D.: Designing software for ease of extension and contraction. IEEE Transactions on Software Engineering, (March 1979). 128-138
16. Meyer, B.: Applying Design by Contract. IEEE Computer, Vol. 25, Iss. 10, (1992). 40–51
17. Cysneiros L., do Prado Leite J.: Non-Functional Requirements: From Elicitaion to Modelling Languages. International Conference on Software Engineering (2002)
18. Elrad T., Filman, R., Bader A.: Aspect-oriented Programming an Introduction. Communications of the ACM, Vol. 44, Iss. 10 (2001)
19. AspectJ, www.aspectj.org. (April 2003)
20. Wald, E., Stammers, E.: Out of the Alligator Pool: A Service-Oriented Approach to Application Development. EAI Journal (March 2001)
21. Yang, J., Papazoglou, M.: Interoperation Support for Electronic Commerce. Communications of the ACM, Vol 6, Iss. 43 (2000). 39-47
22. P. Johannesson, B. Wangler, P. Jayaweera,: Application and Process Integration - Concepts, Issues, and Research Directions. Information Systems Engineering Symposium, Springer Verlag (2000)
23. Channabasavaih, K., Holley, K., Tuggle, E. M.: Migrating to a service-oriented architecture, Part2. IBM developerworks, http://www-106.ibm.com/developerworks/webservices/library/ws-migratesoa2/, (December 2003)
24. Herzum, P., and Sims, O.: Business Component Factory. OMG Press (2000)
25. Cheesman, J., and Daniels, J.: UML Components. Addison-Wesley (2001)

Paper D

Moving from Internal to External Services using Aspects

Martin Henkel, Gustav Boström and Jaana Wäyrynen

Proceedings of the First International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA'2005), Springer Verlag, Geneva, Switzerland, February 23 – 25, 2005.

Moving from Internal to External Services using Aspects

Martin Henkel, Gustav Boström and Jaana Wäyrynen

Department of Computer and Systems Science
Stockholm University and Royal Institute of Technology
Forum 100, SE-164 40 Kista, Sweden
martinh@dsv.su.se, gusbo@kth.se, jaana@dsv.su.se

Summary. Service oriented computing and web service technologies provide the means to structure an organisation's internal IT resources into a highly integrated network of services. In e-business and business process integration the internal services are interconnected with other, external organisations' resources to form virtual organisations. This move from using services internally to external use puts new non-functional requirements on the service implementation. Without any supporting technologies, meeting these new requirements can result in re-writing or changing a large part of the service implementation. In this paper we argue that aspect oriented programming is an important technique that can be used to facilitate the implementation of the new requirements that arises when moving from internal to external services. The suggested solution is illustrated by an example where quality of service metrics is implemented by using aspect oriented programming.

1 Introduction

Service oriented computing, in particular the web service technologies, has drawn a lot of attention in recent years. The reason for this attention is multi-faceted. One reason is based on the view that service oriented computing is a natural step of evolution from object-oriented and component based computing. Another aspect that makes services interesting is that they can be used to structure and interconnect an organisation's internal IT systems. Even more importantly, services can be used externally to enable interconnection of enterprises [1], thus enabling the forming of networked or "virtual" enterprises [2].

The interconnecting of enterprises via service technology requires that part of an enterprise's internal systems must be made available to external organisations. This shift from internal to external use puts new requirements on existing IT systems. First of all, the systems must be able to communicate, regardless of differences in platforms and languages. This interoperability can be achieved by conforming to technical standards. The existing and upcoming web service standards such as

SOAP [3] and WSDL [4] are important steps towards interoperable services. A second, somewhat overlooked part is that providing external services also puts new non-functional requirements on existing systems, such as scalability, security and quality. These requirements are sometimes also called “illities” [5].

Within the border of a company these requirements might be implicitly met by assumptions about the company’s secure intranet, the well-defined number of users, the support department’s ability to monitor the quality etc. However, exposing parts of the system to external partners will require that these implicit assumptions need to be converted into explicit, measurable, and monitorable implementations. Furthermore, the new non-functional requirements posed need to be implemented as part of the existing systems.

Implementations of new non-functional requirements often cut across the entire system, i.e. a large part of the existing system code is affected by new non-functional requirements. Thus, when moving from internal to external service use, a potentially large part of the system code needs to be changed. The need to change large portions of code to implement new non-functional requirements is a problem, since it increases the risk of major redesign when moving from internal to external use of services. The question is how to overcome this problem.

Aspect Oriented Programming (AOP) is proposed as a technique to implement functionality that cuts across an entire system [6]. Thus, aspect orientation may be a solution that can facilitate the implementation of the non-functional requirements that arises when internal systems are to be exposed as external services.

In this paper we argue that aspect oriented programming can be a useful technique for moving from internal to external services. The paper begins with a short introduction to service oriented computing and aspect oriented programming. The introduction is followed by a description of the problem and the proposed solution. Then, we provide an example of how aspect-orientation can be applied in the context of a service that needs to be changed due to changed non-functional requirements. The article ends with a discussion of the proposed solution’s applicability and a discussion of further work.

1.1 Service Oriented Computing

A *service* is an “act or performance offered by one party to another” [7]. Services offered by IT systems have been dubbed “e-Services” [8] and “software services”. In this paper the term service denotes services offered by one system to another system, i.e. where both parties are software systems. This excludes services that are offered to (human) users via graphical user interfaces.

Service Oriented Computing (SOC) builds on the component based development (CBD) principles of building systems by combining software parts. In contrast to components, a service is a run-time programming interface rather than a physical/binary entity that needs to be installed before use. This distinction between runtime provisioning and implementation is made by component based development methods [9]. The distinction might seem finicky, but it has a profound impact on how services are used. A provider of a service is responsible for the run-time availability of the service, whereas a component provider is only

responsible for the construction and delivering of the binary component. Thus, building a component based system is about assembling software parts, while building a service-oriented system is about communicating with services offered by different providers.

The focus on run-time availability and provider responsibility makes services an ideal metaphor for interconnecting a organisation's IT-systems (internal use), where each system is a separate run-time entity. For the same reason, service oriented computing can play a major role in the interconnection of systems belonging to separate organisations (external use).

1.2 Aspect Oriented Programming

Aspect oriented programming is a paradigm that attempts to help in implementing concerns that are present in many modules and therefore crosscuts a system, that is *cross-cutting concerns*. Cross-cutting concerns are difficult to modularise using existing object-oriented techniques since there is no logical place in which to implement them. An illustrative example is logging of method-calls. Using existing object-oriented techniques the code for implementing this would be spread out in all methods that require logging. Changing the way logging is done, and especially, where it is performed is therefore difficult to accomplish without changing all methods that need to be logged. This poor modularisation leads to code that is difficult to maintain. The fact that you need to deal with several concerns, logging and business logic, in the same method also adds to the complexity of the code.

Aspect oriented programming provides a way to modularise these cross-cutting concerns in an efficient manner by factoring out logic belonging to a specific concern into an *Aspect* [10]. In this article we use AspectJ as a tool to implement aspect oriented programming [11]. An aspect in AspectJ consists of *Pointcuts*, and *Advice* (in AspectJ an aspect can also contain *Inter Type declarations*, but this concept is not used in this article). *Pointcuts* describe *where* the aspect should apply in terms of the object-oriented systems structure, e.g. the pointcuts of the logging aspect would describe *where* in the system logging should be performed in terms of the classes and methods of the system. *Advice* describes *what* should happen at these pointcuts, e.g. how the logging should be carried out. The AspectJ keywords *aspect*, *pointcut* and *advice* are added to the Java-syntax in order to support these concepts. The process of combining the aspects and the classes into an executable system is called *aspect weaving*.

1.3 Functional, non-functional and cross-cutting requirements

Functional requirements are statements of services that the system should provide. This can also be said as describing *what* the system should do. Non-functional requirements, on the other hand, are focused on *how* the system should perform the services [12]. An example of a functional requirement on an ATM-machine could for example be that an ATM-machine should be able to dispense money to the bank's customers. A non-functional requirement could be that this service has to be performed *securely* and with an acceptable *response time*. Other examples of non-

functional requirements are performance, traceability, scalability and error handling. A problem with non-functional requirements is that they are often *crosscutting*, i.e. they affect many modules of the system. For example, security needs to be addressed in many parts of an ATM-system. It is therefore difficult to modularise crosscutting requirements. This can make systems difficult to maintain and evolve [13].

2 Moving from Internal to External Services

As stated in the introduction, service oriented computing promises to interconnect organisations. This is done by integrating and automating business processes that span across several organisations. Service oriented architectures (SOA) and service technologies, such as web services form the fundament for such integration. Integrating business processes and automating them relies on the integration of the organisation's IT systems. This in turn, requires that contact patches between the systems need to be established. Systems, which previously only where used within a company, need to exchange information with external systems through these contact patches. Interconnecting the processes of two organisations commonly does not require that all the IT systems need to be integrated. Rather than making an entire system available externally, a selection of functionality is made. This functionality is then exposed as services that can be used by external organisations [14]. As mentioned earlier, the exposed services commonly need to adhere to a new set of non-functional requirements. Examples of new non-functional requirements are security, quality of service measurements, and performance monitoring. Implementing support for these new requirements is instrumental in making the services available externally.

There exist several solutions to this problem. The first solution that comes to mind is to rework the entire code to support the new requirements. This can be achieved by following common refactoring principles, such as those proposed by Fowler [15]. However, this solution requires a lot of work, since each method that should adhere to the new requirements has to be reviewed. More generic approaches have also been proposed, such as the addition of an extra layer to existing component-environments by using generated proxies controlled by proprietary description languages [16]. These generic approaches have a much better chance of reducing the amount of work required. They are, however, based on proprietary languages and technologies. The ideal would be a technique which is generic (to avoid too much rework), and at the same time does not rely on proprietary technologies, servers, and languages. We propose that aspect oriented programming can be such a technique.

Aspect oriented programming separates the code required to fulfil the new requirements from the existing code. The new requirements can thus be separately implemented as aspects, without changing the existing code. These aspects can then selectively be applied (by aspect weaving) to the parts of the code that need to adhere to the new requirements. Applying the aspects does not require changing the original code. For a large system this can save a lot of time.

In the next section we will give an example of how aspect oriented programming can be used to implement non-functional requirements without a major rework of the original system.

3 An example: Adding QoS Metrics to Web services

The example in this section describes the steps necessary to extend a web service with quality of service metrics monitoring (QoS monitoring). The example elucidates the main point of this paper, that by using aspects, the move from internal services to external services does not require a major rework of the code. The need to extend web services with QoS metrics is selected as an example both because it is a likely scenario, and because it clearly demonstrates how non-functional requirements can be implemented using aspects. What makes the scenario likely is that enterprises starting to use the web service technology internally will need to further define, and monitor their quality of service when starting to use web service technologies as an external communication mean between enterprises, thus the need to add QoS metrics to web services.

3.1 Scenario

To illustrate how AOP will help in implementing non-functional requirements such as QoS metrics let's imagine a company that provides financial services such as mortgages and loans for cars. In this business it is essential to know your customers' credit worthiness. Credit worthiness is determined using the customers' credit history, income and other variables. Different financial services require different definitions and levels of credit worthiness. This information is used in determining whether to grant applications for both loans and mortgages. Since credit checking is an important part of this organisation's business, it is implemented as a web service that can be reused from all systems within the organisation. Figure 1, below, shows how the interface to this credit checking service might look like implemented in Java.

```
public interface CreditCheckingServiceInterface
{
  public boolean hasPaymentRemarks(String name);

  public boolean hasCreditHistory(String name);

  public boolean checkCreditForAmount
                  (String name, int amount);
}
```

Fig. 1. The interface of the CreditCheckingService

The next step in the organisation's business plans could be to provide the credit checking service to external businesses, such as mobile phone operators and car leasing companies that also need efficient credit check processing. However, before using the credit checking from their systems, external businesses will require some form of quality guarantee. For example, a potential customer of the service would probably ask the following questions:

- How can it be ensured that the service paid for is reliable and running when needed?
- How can the organisation monitor that the performance is acceptable?

In short, the customers will require some form of agreement that states the intended quality of service. The agreement can include measurable limits for performance, cost, up time and other dimensions that affect the overall quality of the provided web service. For this example we use three QoS dimensions for web service processes as defined by Sheth et al. [17]: time, cost and reliability.

- Time is a measure of response time of the web service that is to be monitored. The response time is measured from request arrival to the completion of the request.
- Cost can be measured by either estimating an average cost for each service invocation, or by measuring the resources that are consumed to complete a request (such as processor time, cost of information storage etc).
- Reliability is a measure of technical failure rate, that is, monitoring the reliability will discover how many times the service failed to deliver a response. Sheth et al. [17] suggest that reliability is to be measured as a ratio of successful executions/scheduled executions.

The credit checking web service mentioned above is not built with QoS metrics in mind, since it was designed for internal use only. Adding QoS metrics to the existing service can be a major undertaking, since code that monitors the metrics need to be inserted in all parts of the service. Without a technique that helps implement cross-cutting, non-functional requirements such as QoS metrics, developers are running the risk of having to redesign a major part of the code. However, applying aspect oriented programming can reduce this risk. An example of how aspects can be applied in this case is described in the next section.

3.2 Applying Aspects

Let's look at how aspects could be applied in the described scenario. The three QoS dimensions time, cost and reliability define what is to be measured. Before implementing the actual metrics, it has to be decided where in the application code the dimensions should be measured. A basic approach would be to add code to

register each metric in the beginning and end of each request, i.e. before and after each call to the web service. Without using aspects, this approach would require additional code that has to be inserted in *all* web service methods. However, using an aspect-oriented approach, adding QoS metrics to web services would only require the metrics *aspects* and their *join points* to be defined once, without any change to the original web service implementation.

To implement QoS metrics for the credit checking service, one aspect for each of the QoS dimension can be implemented. Thus, as an example we have implemented the aspects PerformanceQoSAspect, CostQoSAspect and ReliabilityQoSAspect .

```
public aspect PerformanceQoSAspect
{

   Timer timer=new Timer();

   pointcut timedMethods() : (
       execution(public * CreditCheckingService.* (..)));

   before() : timedMethods()
   {
      // Start timing
   }

   after() : timedMethods()
   {
      // End timing
   }
}
```

Fig. 2. Performance QoS aspect

3.3 Performance Aspect

The performance aspect is intended to measure the Time QoS dimension. Time can be measured by recording the request/method name, when the request arrived and when the response was sent. The implementation of this metric requires that two aspect join points are defined; one at the beginning of each method call and one at the end. These join points are defined within the AspectJ pointcut “timedMethods”, see figure 2, above.

3.4 Cost Aspect

Cost can be measured by recording the request/method name for each request. Using predefined cost for each type of request, the total cost can be calculated. The

measurement of cost can be done by using a join point at the end of each web service method. The AspectJ example in figure 3 shows how an aspect that logs each method call can be implemented.

```
public aspect CostQoSAspect
{

   pointcut costMethods() : (
       execution(public * CreditCheckingService.* (..)));

   after() : costMethods()
   {
       // Log the cost of the executed method
   }
}
```

Fig. 3. Cost QoS aspect

3.5 Reliability Aspect

Reliability can be measured by recording if the response of a request is a valid response or an error. In this case, a join point can be defined at the end of each method. The AspectJ implementation shown in figure 4 defines an aspect that logs every method call that ends with a non-application Exception.

```
public aspect ReliabilityQoSAspect
{
   pointcut reliabilityMethods() : (
     execution(public * CreditCheckingService.* (..)));

   after() throwing(Exception e): reliabilityMethods()
   {
     if(!(e instanceof ApplicationException))
     {
       // Log error
     }
   }
}
```

Fig. 4. Reliability QoS aspect

The example given above can be extended with more QoS metrics. This example illustrates the main points in using aspects for the implementation of non-functional requirements.

4 Conclusion

In this article we proposed that AOP could help the transition from internal to external services. By using AOP, non-functional requirements can be implemented without doing a major redesign of the existing system. The feasibility of the proposed solution has been demonstrated with a simple example written in AspectJ.

The example demonstrated that AOP could be a useful tool when an application needs to accommodate QoS metrics that have not been previously designed into the system. It also shows that this can be easily achieved using just a few lines of code. In fact, the bigger the application, the more amount of time will be saved by using aspects.

AspectJ was used in the example. However, there are other ways to implement non-functional requirements in an "aspect oriented" way. It could be argued that by using a component technology such as Enterprise Java Beans (EJB), QoS metrics could be provided by the application server (e.g. through the use of method interceptors in the JBoss EJB server [18]). These QoS metrics, however, are not currently standardised, they would therefore be different for each component server. It would also require the application to be built as a component-based application from the start, which is often a lot more time-consuming and skill-intensive than using plain Java objects. Using design patterns such as the "proxy" pattern [19] could also alleviate the need for using specific AOP technologies such as AspectJ. An example of this is provided by Filman et al. [5]. This approach, however, is considerably more time-consuming and therefore also more error-prone.

The proposed solution is applicable when the move from internal to external services poses new non-functional requirements. Clearly, if no additional non-functional requirements need to be fulfilled, the need to introduce aspect-oriented concepts is not as obvious. Furthermore, the proposed solution presumes that the non-functional requirements can be implemented in a generic, separated way using aspects. In the case that not all new requirements can be implemented in this way, aspects can still contribute to the implementation of some of the requirements. Thus, we believe that the use of aspect-oriented programming can be a valuable technique when moving from internal to external services.

5 Further work

In this paper we examined how aspect oriented programming can be used to tackle the non-functional requirements when moving from internal to external services. However, when integrating processes it is likely that other changes need to be implemented in parallel with the new non-functional requirements. For example,

when integrating processes there might be a need for further process automation, i.e. new functional requirements. A possible future extension of our work might include principles guiding the combination of aspect-oriented programming with traditional refactoring techniques to implement both non-functional and functional requirements.

Another interesting question is whether AOP could prove useful for solving other "architecture breaking" problems. There are several indications that this could be the case. De Win et al. [20] have shown how AspectJ can be used to help implement security features in an application. Filman et al. [5] have described how AOP can be used for inserting "ilities", such as stability and reliability. These examples, however, do not prove that AOP can handle every possible new requirement.

References

1. Fremantle, P., Weerawarana, S., Khalaf, R., Enterprise Services. Communications of the ACM, October 2002, Vol. 45, No 10 (2002)
2. Yang , J., van den Heuvel, W. J., Papazoglou, M. P., Service Deployment for Virtual Enterprises. Australian Computer Science Communications, Vol. 23, Iss. 6 (2001)
3. Gudin, M., Hadley, M., Mendelsohn, N., Moreau, J., Nielsen, H. F., SOAP Version 1.2. W3C Candidate Recommendation. (2002)
4. Chinnici, R., Gudin, M., Moreau, J., Weerawarana, S., Web Services Description Language (WSDL) Version 1.2. W3C Working Draft (2003)
5. Filman R., et al., Inserting Ilities by Controlling Communications. Communications of the ACM, Vol. 45 (2002)
6. Duclos F., Estublier J., Morat P., Describing and Using Non Functional Aspects in Component Based Applications. 1st International Conference on Aspect-Oriented Software Development (2002)
7. Lovelock, C., Vandermerwe, S., Lewis, B., Services Marketing. Prentice Hall Europe (1996)
8. Piccinelli, G., and Stammers, E., From E-Process to E-Networks: an E-Service-oriented approach. OOPSLA Workshop on Object-Oriented Web Services (2001)
9. Allen, P., Frost, S., Component-Based Development for Enterprise Systems: Applying the Select Perspective. Cambridge University Press (1998)
10. Elrad, T., Filman, R., Bader, A., Aspect-oriented Programming an Introduction, Communications of the ACM Vol. 44 (2001)
11. AspectJ, www.aspectj.org. Accessed in April 2003
12. Cysneiros, L., do Prado Leite, J., Non-Functional Requirements: From Elicitaion to Modelling Languages. International Conference on Software Engineering (2002)
13. Moriera, A., Araújo, J., Brito, I., Crosscutting Quality Attributes for Requirements Engineering**.** 1st International Conference on Aspect-Oriented Software Development (2002)
14. Georgakopolos, D., Schuster, H., Cichocki, A., Baker, A., Managing Process and Service Fusion in Virtual Enterprises. Information System Vol. 24, No6 (1999) 429-456
15. Fowler, M., Refactoring – Improving the Design of Existing code. Addison-Wesley (1999)
16. Becker, C., Geihs, K., Quality of service and object-oriented middleware - multiple concerns and their separation. International Conference on Distributed Computing Systems (2001) 117 -122

17. Sheth A., Cardoso J., Miller J., Kochut K., Kang M., QoS for Service-Oriented Middleware.. Proceedings of the Conference on Systemics, Cybernetics and Informatics, (2002)
18. JBoss, www.jboss.org, Accessed in April 2003
19. Gamma E., Helm R., Johnson R., Vlissides J., Design Patterns – Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
20. De Win, B., Vanhaute, B., De Decker, B., How Aspect-Oriented Programming Can Help to Build Secure Software. Informatica Journal, Vol. 26, (2002) 141-149

REPORT SERIES NO. 06-015
ISSN 1101-8526
ISRN SU-KTH/DSV/R—06/15--SE

Department of Computer and Systems Sciences
STOCKHOLM UNIVERSITY, SWEDEN