Abstract

Service-oriented architectures focus on the alignment of business processes and the supporting information technology. To facilitate the alignment and to decrease administration costs we propose linking of SOA services to enterprise models respectively target state organisational models by using semantically enriched conceptual modelling languages. Acting on language meta-level means in this context the introduction of service related constructs into widely used visual modelling grammars. Vision is the situation dependent adaptation of software functionality to the actual business process requirements. An efficient adaptation is crucial in order to be able to respond quickly to environmental or system induced change.

Keywords: Enterprise Architecture, Service-oriented Architecture, Enterprise Model, Web service.
1 INTRODUCTION

Due to increased competition and shortened time-to-markets, companies gained significant resources for quality management in the recent years. Simultaneously a strengthened process orientation is indicated within the whole commercial scope (Maier and Remus, 2001). Due to the intention of quality improvements but also to the increasing cost pressure through globalisation, the focus of rationalisation is more and more related to the efficiency and effectiveness of its business process structure (Kettinger, 1997). Prerequisites for an effective intervention into the process domain are in the first instance the entire analysis of the problem domain and its consistent documentation in the course of business process reengineering activities (Hammer, 2003). Within this scope of first stage analysis and documentation, visual models are established – so called enterprise models representing business requirements for the actual enterprise situation (Fettke and Loos, 2003). Corresponding modelling languages provide a graphical notation for high-level semantic description of real world phenomenon.

While enterprise models are an established technique for business analysis and development, the complexity of enterprises requires a more holistic view. Understanding enterprises as complex information systems, there are not only business aims and business processes but also software and data. For a successful development of an enterprise, it is necessary to integrate these different views. Enterprise architectures (EA) have been established to fulfil this task. The definition of EA varies (Beznosov, 2000) but is often understood as a framework that “…provides a way of viewing a system from many different perspectives and showing how they are all related.” (Sowa and Zachmann, 1992) Thereby, enterprise architecture should not been mixed up with software architectures, that are seen as the “…structure of the components of a program/system…” (Garlan and Perry, 1995).

While enterprise architectures can help understanding and managing the different perspectives, they generally do not solve problems arising when one view (or architectural layer) alters. In that case, the other views have to be adapted according to the dependencies existing between them. That means, when an enterprise model (e.g. process model) changes, the software supporting that process has to be tailored. That is usually carried out by functionality adaptation or by a complete new implementation. However, that will lead to exceptional effort, which is not possible to be justified in line with a reduced budget caused by increasing cost pressure. Possible financial savings through convenient process improvements are thereby scattered due to the adaptation of associated application systems.

To avoid that manual software adaptation, we suggest an automated binding of functionality (IT) to the described enterprise models. Therefore, we propose a scenario, where business-supporting software directly accesses the models to adopt its behaviour according to the described processes, organisational responsibility or whatever is stored within the enterprise models. We can identify three tasks for an appropriate realisation of our vision:

1. Bridging the gap between analysis and design models.
2. Providing real-time access to relevant information of the enterprise models.
3. Development of flexible business-supporting software accessing that information.

To ensure real-time access to the enterprise models, their content should be available electronically. Due to the raising usage of modelling tools, this is not a problem in modern enterprises. Thus, a connection between the business software and the modelling tool (or the modelling tool data) is required. For such a connection, the SOA paradigm (Erl, 2005; Stojanovic and Dahanayake, 2005) offers interesting methods of resolution. Due to their advantages and potentials in the present case, the focus lies on the W3C’s Web service standards.

In this paper, we present a grammar to model such Web services. Thereby we do not focus on a general modelling approach, but focus on the enrichment of existing enterprise models with Web service elements. To realise a solution for enterprise models in different modelling languages, our approach aims at the meta-model level. In addition to the modelling approach, we present a framework that illustrates how such enriched enterprise models can be used for the scenario presented above.
The paper is structured as follows. In the next section, we give a brief survey about requirements for conceptual modelling of SOA. In chapter 3 we introduce the E³ plus method that shows how enterprise and Web service models can be integrated using object patterns embedded in the method engineering field. The paper ends with a discussion, summarizing the proposed ideas and exposing open questions regarding the realization of the application integration.

2 REQUIREMENTS FOR CONCEPTUAL MODELLING OF SOA

In SOA management we distinguish between management of Web services and management by Web services (Alonso, 2004). Management by Web services references the integration of heterogeneous, distributed systems due to Web service’s ability to reduce heterogeneity with standardised languages and protocols. In contrast to that, management of Web services concerns the monitoring and control of the Web services themselves and their runtime environment to assure that they correspond in the agreed quality (Casati, 2003).

The present paper focuses on the management of Web services whose scope is divided into the infrastructure level, the application level and the business level (Casati, 2003; Curbera, 2003; Sheth, 2006). The first two are mainly related to performance issues and Web service life cycles – subjects that are common practice in traditional application and network management, too (Hewlett-Packard, 2003; IBM, 2002). By contrast the business layer addresses the real new aspects that come along with the SOA paradigm. As Web services are intended for B2B transactions they usually have a higher level of abstraction compared to traditional middleware objects like CORBA (Casati, 2003). Issues are executable business processes through service composition, Quality of Service (QoS) information and measuring the value of IT investments by transparency of IT operations from a business perspective (Casati, 2003).

We concentrate on two aspects:

- Firstly, the estimation of effects of a service execution (compounded and atomic).
- Secondly, the adjustment and optimisation of service executions according to the business objectives.

The present modelling purpose is the model-driven configuration of service compositions at business level. Below we give a brief survey about the findings of a theory driven hypothesis exploration. A literature review combined with practical experience provides a basis for the construction of a requirements model representing the main assumptions for a successful modelling approach. The assumptions are arranged according to the general phase model of the systems engineering process (Balzert, 2001).

2.1 Analysis phase

The analysis phase includes the problem definition and the environmental analysis of the system under development. Though, the focus lies at the deduction and formulation of requirements to the software system – describing it from an outside perspective (Balzert, 2001). To get into specific problem domains the modelling approach is an important technique gaining essential benefit through the reduction of complexity by abstraction which facilitates analysis of complex systems (Balzert, 1994; Ferstl and Sinz, 1994). Models can be understood as the result of a construction "... done by a modeller, who examines the elements of a system for a specific purpose ..." (Schütte and Rotthowe, 1998). Hence, enterprise models describe mental representations of real world phenomena with constructs representing structural (e.g. things and their properties) as well as behavioural aspects (e.g. events and processes). A method to create enterprise models consists of a modelling grammar (language), providing a set of elements and rules to combine the elements and a procedure by which a grammar can be used (Wand and Weber, 2002). For further scientific analysis, we will use enterprise models as synonyms to conceptual models.

The construction of enterprise models often occurs within the requirements engineering during the information system development (Wand and Weber, 2002). The semantic mightiness of their modelling language elements has to cover non-formal aspects supporting a deep understanding of the business domain and of the potential of an information technology employment as well as formal
aspects in order to support the system implementation (Frank, 1999). Thus, we use semi-formal languages to model problems which are not well-structured, highly subjective, individual and finally not objectively well formed (Harel and Rumpe, 2000). The present modelling purpose implies the transformation of the problem description modelled with semi-formal languages into a shape that allows an appropriate automatic processing.

Three general assumptions can be derived from the given modelling purpose:

**A1:** To realise such a processing, the problem has to be described in a formal shape (Krämer, 1988). Prerequisite for a formalisation of the transformation is the completeness and correctness of the problem description. Thus, if enterprise models should serve as input into a formal transformation, all required information for the specification of the software system must be gathered in the requirements model. Missing or incorrect information increase the degree of freedom in the software design and thus the likelihood of the problem inadequateness. Hence, an automatic transformation demands a complete input containing all information relevant to the transformation. Though, the completeness of enterprise models cannot be subject independently assured. Furthermore, it is impossible to evaluate the correctness or significance of modelled information.

**A2:** Formally, the modelling purpose at hand requires an obligatory interpretability of functional and non-functional requirements models through an automatic machine. This implies the use of a formal language for requirements modelling (Krämer, 1988). As the deduction of requirements from a problem description is the result of an inter-subjective process that is based on communication, the use of a formal language during the requirements analysis causes a shortened understandability of the models and a greater likelihood that requirements are not fully or properly implemented. Thus, enterprise models lose their role as artefact in a process of consensus-making. In addition, non-functional requirements are not able to be quantified and formally evaluated (Karow and Gehlert, 2006).

To sum up, recent research shows that enterprise models in their role as problem description serve as input for a creative, manual decision process that produces a solution in terms of a software design (Génova, Valiente and Nubiola, 2005; Karow and Gehlert, 2006). The main problem in conventional enterprise modelling approaches is the use of semi-formal languages which share strength and weaknesses with natural languages. As a result of a high degree of freedom within the modelling activity, we have naturally no standardised set of modelling elements. This will lead to a lack of comparability that becomes an issue as we need to configure distributed construed design models of SOA services within one single enterprise model.

**A3:** Thus, we assume that the establishment of linking between organisational and IT domain can only be done with the provision that we use a set of entities simultaneously in both domains (Juhrisch, 2008).

### 2.2 Design and implementation phase

Design and implementation phase are meshed together transforming the program design into executable code. With regard to the increased effort concerning a model driven software engineering, we disavow a separate treatment of both phases in the present paper.

While enterprise models describe the task of the software system from an outside perspective, models construed during the design phase represent the software system itself or parts of it (Evermann, 2005). This model class is concerned with the question how the system implements its tasks. Therefore, the artefacts of the analysis phase are used to design the structure of the software system that is made up of system components and their relationships among each other – the software architecture (Balzert, 2001). The system component is the main entity of the design and the implementation phase.

A service-oriented architecture provides the basis for the present software architecture (W3C, 2003). Services constitute the interfaces of system components (Humm, 2006). They represent a defined business functionality that is exported by the component. Service functions have the ability to be reusable in more abstract processing sequences. Their signature is defined in terms of the input respectively output parameters and corresponding types.
Unlike to the modelling activities in the analysis phase, decisions in the design phase are mainly implementation driven – basically affected by the development paradigm. Components have to be identified in a process of system decomposition; their services have to be designed. For the proposed approach we assume that the given SOA has reached a stage of development where the premises for a reuse of software solutions in an organisation are complied. Representative requirements are a uniform minimum quality of service specification; service functions that comply with the design by contract requirements; well established development procedures for reuse oriented service design; services with a high cohesion, weak dependencies to other services and the entire definition and uniform representation of business objects (Dietzsch and Goetz, 2005; Humm, 2006; Keen, 2004; Lim, 1998).

Since with a service we do not possess transparency about the semantic of its functions, the SOA expatiates itself primarily by the amount of available service interfaces. To operate in a SOA, a Web service has to declare its functional and non-functional requirements and characteristics in an agreed, machine-readable form (Curbera, 2003). Web services provide standards for the elimination of heterogeneity at platform- and system and integration level. However, it is crucial to solve heterogeneity in differences in data schemas at semantics levels and in functional descriptions, too. SOA management requires a business access to a Web service in order to estimate its business value and its compatibility in a business scenario (Krafzig, Blanke and Slama, 2004; Sheth, 2006).

Based on the general assumptions above, we derive two main requirements to the process of conceptual SOA modelling.

R1: The main requirement for this language constructs represents the avoidance of any serious restriction to the ordinary modelling task at business level. Usually the construction of enterprise models requires “a specific competence that is neither covered by software engineering nor by organization science” (Frank, 1999) so implementation related information has to be kept away. The main problem from a modelling view is the common use of language constructs originally intended exclusively for the software design. Thus, design decisions may exert influence on the way the analysis of the organisational domain is modelled. To avoid this, the inserted modelling elements must be basing upon a consensus between domain experts and software engineers.

R2: Service composition becomes the essential issue within the design phase (Curbera, 2003). The orchestration of Web services in a specific order gets aligned to accomplish a certain business goal. Business processes describing the steps to achieve a business goal are modelled in computation independent enterprise models. In case the Web services should be invoked in the order of their equivalent functions in a business process, we have to automatically generate orchestrations or choreographies of Web services directly out of enterprise models. Hence, business processes have to be modelled in a way that allows an automatic transformation into executable orchestrations. Essentially, the decision on task automation has to be identifiable in the requirements model. For it, Web services have to be bounded to business functions. Since this binding is only achievable via a semantic description of functional and non-functional characteristics of the services and the business function, the Web service has to be modelled, too. We require a mapping between enterprise models and the service models characterized as design models.

To sum up the recent section we can say that the business orientation is the main difference to previous software component approaches within the software engineering discipline (Busi, Gorrieri, Guidi, Lucchi and Zavattaro, 2005). The present approach seizes this idea and introduces a modelling framework carrying out a business access to Web services. This implies the extension of existing enterprise modelling languages with concepts that aim at bypassing the gap between analysis and design phase of systems engineering process. Therefore, we access the grammatical specification of conceptual modelling languages. Acting on language meta-level means in this context the introduction of Web service related concepts into widely used visual modelling grammars.
3.1 Meta-modelling using Object Patterns

The use of language in systems engineering is split into two phases: language creation and modelling. Language creation references the description of the syntax and semantics of a modelling language with recourse to a meta-language that in turn can be a modelling language. If so, we talk about a meta-model (Frank, 1999). Now, to overcome the epistemological gap between semi-formal problem description and formal software solution, we propose the modelling approach E³plus in the language creation phase. If a modelling language is defined on meta-model-level, we integrate restricted language constructs whose implementation is guaranteed within the SOA.

As concepts that matter at implementation level in software engineering but also at the level of conceptual modelling we recommend a standardised set of patterns. Object patterns (OP), attribute patterns (AP) and attribute value patterns (AVP) are used as a framework to describe business objects or parts of business objects in models. While the object pattern itself is the collection of all the attribute patterns that may be needed in a concrete instance of that pattern, a concrete instance describes a concrete business object or part of it. A concrete instance of an OP consists of concrete instances of APs. To describe the state of a business object used in a model, one can add (concrete instances of) AVPs to the APs of an OP and combine them by logical operators.

If a pattern is used in a enterprise model, it concretizes the conception of a modeller about a business object on a purely linguistic level. In contrast, in design models the instances of patterns can be attached with concrete names, as they are used for data types processed in the described Web service function. By this allocation of names (respectively concrete values in the case of AVPs) a mapping can be produced, e.g. by a XSLT sheet, in automated way which is used by the Enterprise Service Bus (Luo, Goldshlager and Zhang, 2005) – the central architectural control of the SOA – for a transformation.

Thus, by using the same patterns in both business and software design models we can bridge the semantic gap between organisational and IT domain (see Figure 1).

![Figure 1. Modelling with the E³plus method](image)

The method engineering is based upon the existing E³-model (E³; Greiffenberg, 2004) classified as meta-meta-model of the Meta-Object-Facility Architecture (OMG, 2002). A meta-model of a modelling language is composed of concepts – the vocabulary – and rules for their combination. The modelling of OP takes place in the meta-modelling process, too (see Figure 1). An OP results from the assignment of APs. Elementary APs are e.g. “Name”, “Removal Date” or “Matriculation number”. In addition, APs that represent discrete entities are modelled using AVPs. An example is the AP “Role”. Assigned AVPs are e.g. “student”, “employee” or “associate professor”. The extra work to predetermine some AVPs follows from the necessity to explicitly specify a business object state. Lately, E³plus desists from the entire modelling of all possible AVPs because that would require a complex body of rules. However, corresponding considerations are subject of further examinations.

To sum up, an OP represents a framework for the description of one or many business objects. This frame results from a consensus of domain experts and system engineers concerning the vocabulary to describe certain entities. As a consequence, the patterns build grammatical constraints for the
description of entities in enterprise models and design models so that they can close the gap between these two modelling domains.

In contrast to the Business Objects (Weske, 1999) the modelling of OP does not aim at companywide uniform business objects but rather represents the consensus between software architects and domain experts regarding the language to describe business objects.

3.2 Conceptual modelling using Object Pattern Instances

When typing the meta-model all elements and their relationships are implemented in the meta-CASE tool and are made available for the enterprise modelling activities. Hence, we use instances of OP – concrete object descriptions – in enterprise and design models. An OP instance is one of all possible modelled conceptions of a business object within the predetermined OP.

3.2.1 Service function model

With E’plus a model for SOA was developed that pictures a service interface with the help of graphic characters and interlinks method signature to instantiated object pattern (see Figure 2).

Figure 2. Service model

At message parameter definition object patterns come into action. Thereby, a concrete object is described by allocation of AttributePattern instances into ObjectPattern instances. For detailed information about the notation Juhrisch is referred (Juhrisch, 2008). We distinguish between two types of service functions:

- Atomic service functions that can be allocated exactly to one business object or technical concept and the semantic results in the status change of the object. These service functions do not base on the other service functions. It is assumed that the semantic of these functions expatiates on the input- and output objects that would certainly work out in data processing.

- Furthermore service functions exist where the semantic only in the detail level of the functional description results. Thereby refined service functions can consist of other service functions or also from functions that do not have a service character. The concept of complex service functions in an SOA model results in status transfer models and it orients itself towards the principles of coupling and cohesion.

An OP is instantiated and forms one concrete object description through the assignment of AP and AVP instances. The connection to the service messages is realised via aggregation edges. Figure 2 illustrates the modelling of a person as input respectively output object of the service functions “removeStudent()” and “registerAlumni()”. The necessity to attach instances of AVP to the AP instance “Affiliation” and “Status” follows from the necessity to explicitly specify, what a service function changes in the business object state. Invalid state-transitions have to be avoided. For more
detailed information about the modelling notation we reference the work of Weller et al. (Juhrisch, 2008).

The creative decision process during the modelling of services becomes manifest in a XSLT-transformation sheet. This sheet is afterwards attached to the particular services in the ESB environment. There it enables the syntactical compatibility of orchestrated services. Thus, a service does not necessarily need to deal with OP instances, but we assume that the data types can be modelled by OP instances.

Every adjustment of OP on meta-level implies possibly a changed mapping between instances of OPs and the service’s data types. Though, an enlargement of OPs can definitely be done without changing the instances.

### 3.2.2 State-transition diagram

For the model driven configuration status transfer models are introduced that - based on an expansion of event-driven process chain (EPC) - describe functional requirements (Juhrisch, 2008). Following Rosemann (1996) a semantic relationship between event and OP instances is produced and interpreted as an associated operational status that expatiates on collection of statuses of the associated objects.

Thereby, the automatable business logic is documented as an amount of status transfer of objects. In the models of status transfer the orderly procedure of a processing is to organise. With it service functions from the service models can be integrated within the procedure. The service function of a web service is suitable for a certain operational context if the releasing respectively resulting event(s) of a function encompasses OP instances with a status that complies with the service function in the model. The configuration automatically chooses a suitable service function and recommends to the modeller an adaptation of the target model if necessary. As a result the EPC model is ideally balanced with the present SOA. An adapted target model can be transferred as a reference model - a technical solution with an underlying implementation - to other organisations.

### 4 DISCUSSION

Based on the presented modelling approach, we derive four controversial issues regarding the feasibility of the framework.

1. First of all, the entire task to specify OPs (and APs, AVPs) on meta-level and the use of their instances in enterprise models is up to different persons. The modelling of an OP on meta-level demands a process of coordination of persons, who are on the one hand responsible for the method engineering – the method engineers – and on the other hand for the software design – the software engineers (see Figure 3). A consensus is necessary in order to become clear, what sorts of modelling frames - in terms of patterns - are needed; thus, in which domain one wants to model.

The outcome of this consensus could be a bridge between language concepts in enterprise models and the technical aspects in design models. Hence, the presented approach aims for a model driven commitment regarding the employed business objects so that “both sides” use the same vocabulary.
I2: What is still missing is a description what functional features a certain service function is carrying out inside the models. Possibly we can more precisely specify the status of enterprise models by dint of enrichment with state models through OPs, APs and in particular through AVPs. At this point, the presented approach stops short of enabling an exact modelling of business object states using instances of OPs. Furthermore it is not clear – even if we can specify all potential states of a business object – how to model service functionality in a fashion that allow a precise understanding of how a service function processes its business function. The E³plus approach surely works well in situations where the functionality of a service functions can be identified through the states of input and output objects – what is feasible for data processing functions. However, the functionality of a service with “Person” as input and no output is not transparent even if we explicitly describe the “Person” object.

To constrict the semantic heterogeneity we can chop service functions up into almost atomic functions. In turn, this leads to atomic input and output objects what contradicts the fundamental idea of SOA – namely to provide service functions with a granularity compatible to business functions and not just basic technical functions. At this point, the presented approach makes just the first move to the vision of automatic mapping of business requirements to web service functions.

I3: How does "passing along" of business objects work when orchestrating several service functions? If a function in an EPC does not need an object but one cannot forget this because eventually the function may need the object later. There exists a general problem of marking concrete objects (in contrast to concrete instances of OPs, i.e. concretizing of description of a business object).

I4: How do we deal with problems like standard-configuration of attributes or the like? For example a student as an instance of the OP person, who is recognize because the user role is marked as "student"? Some functions assume that they will get a student. It means that we can make a mapping of attribute values; thereby we just have to mark the user role (when converting after a person) on "student", although the service function does not need this attribute at all. Lately, these constraints are already modelled through the assignment of AVPs.

5 CONCLUSION

Coping with complexity in information and communication systems belongs to the main scientific objectives of the information systems discipline in the next years (Heinzl, 2001). The reuse of enterprise models by using them to configure service-oriented architectures follows this idea and
comes up to a major challenge of increasing the use of enterprise analysis methods in businesses and organisations (Tissot and Crump, 1998).

We realized a model-driven development of object patterns on meta-level and their instantiation in enterprise and design models. Regarding the configuration of a SOA within enterprise models, the presented E³plus method allows us to integrate Web service functionality in state-transition models and to test on feasibility in terms of a meaningful semantic process chain. Thus, we achieve an ideal reuse of implemented IT solutions as long as the defined modelling language is usable.

However, testing the state of an OP instance is still a problem. To avoid a prohibited status change, the condition has to be – like already possible in our approach – measurable on the basis of a concrete marking of the AVPs. At this point further studies are necessary in order to enable the modelling of complex policies. Therefore, the focus of our future work lies in addition to the evaluation of the first development results on the theoretical derivative of the approach.

References


