A Service-Oriented Extension of the V-Modell XT*

Michael Meisinger
Institut für Informatik
Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany
meisinge@in.tum.de

Ingolf H. Krüger
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093-0404, USA
ikrueger@cs.ucsd.edu

Abstract

The ever growing size and complexity of both technical and business systems requires efficient software engineering approaches to keep development cost under control while still being able to finish development efforts in time with the required functionality and quality. Systematic software and systems engineering approaches help to push the boundary further and leverage the complexity on many different levels. On the one hand, the availability of appropriate models and notations for the systems under development throughout the development cycle and for all levels of abstraction helps to understand and modify manageable views of the system. On the other hand, systematic development processes can provide the harness for successful project execution and for the ability to repeatedly create results that meet the required quality standards and functionality within the budgeted cost and time. In this paper we combine a proven generic project management framework with a methodology for developing complex multi-functional systems. We embed our service-oriented development *approach for reactive systems into the system development process model* V-Modell XT *by providing a modular extension of the V-Modell XT for service-oriented development. We introduce our development approach by means of a running example from the complex control systems domain, the BART traffic controller example.*

1. Introduction

Enabled by wired and wireless communication technologies, traditional business intelligence and technical systems increasingly converge into ultra-large scale (ULS) systems. Examples of ULS systems include avionics, automotive, command and control, as well as telematics and public safety systems, to name just a few. In all these domains, the primary challenge to software and systems engineering is the integration of a wide variety of subsystems, their associated applications, data models and sources, as well as the corresponding processes, into a high quality system of systems under tight time-to-market, budget, security, policy, governance and other cross-cutting constraints.

Modern ULS systems also have a number of other challenging requirements characteristics that all but exclude monolithic software and systems architectures: ever changing business processes, demands at including both legacy and emerging systems as they become available, and the need to cater to changing requirements, are some examples.

These requirements characteristics have led to a high demand for loosely-coupled integration architectures [16]. As a mechanism to achieve coupling, the notion of *service* has attracted increasing attention both in industry and academia. Web services [32, 39] have emerged in the business domain as an attempt at simplifying distribution, publishing, discovery, addressing, and accessing of software functions across the Internet. The increasing complexity of embedded software has led to a similar trend in automotive and similar technical system domains [21, 1].

Intuitively, web services are application programming interfaces (APIs) that project a subsystem's capabilities to the infrastructure via a well-defined Internet address. It is this *deployment* character of service-oriented development that is behind the recent popularity of the service concept; popular open standards such as HTML and SOAP, together with their integration in Integrated Development Environments (IDEs) have made it virtually effortless to expose subsystem functionality in terms of a web service.

However, the essence of a comprehensive service-oriented software and systems engineering approach cannot be deployment-centric alone. Rather, the idea of structuring an ULS system into a collection of services that project ca-

*V-Modell is a registered trademark of the Federal Republic of Germany. Our work was partially funded by the Deutsche Forschungsgemeinschaft (DFG) through the project *InServe*. Further support was provided by the UC Discovery Grant and the Industry-University Cooperative Research Program, as well as by funds from the California Institute for Telecommunications and Information Technology (Calit2). We are very grateful to Massimiliano Menarini for joint work on this subject.

pabilities of (sub)systems carries much farther than what is addressed by the question of how a service – once identified – can be implemented on the infrastructure.

This paper, therefore, attempts a seamless integration of the concept of services into all phases of the ULS systems development cycle: from logical architecture design to implementation/deployment and maintenance. In general, complex system development requires a systematic approach. Development methods and processes, as well as process and capability models have proven to support the development efforts in the aim for better adherence to time and cost budgets, functionality delivered, quality of the results and repeatability of the entire process. All these processes, models and methods have different properties and application areas, for instance, providing detailed software development methodology support, establishing essential project management practices and helping to comply with specific regulations and standards. They range from quite heavy-weight, sophisticated processes to very light-weight, dynamic agile ones. The particular choice depends on many factors and requires competent and wise project management decisions. In general, each process and method should only be applied with prior adaption to the specific project and organization, and checked for effectiveness during the course of a project.

Problem Definition and Solution Overview Because services are projections of capabilities of (sub)systems, they are fundamentally *partial* in nature. It is the composition of a set of services that yields the desired integration architecture. Most established development processes, however, focus on the notion of component as the unit of development and deployment – this results in a poor match with the partial nature of the service concept.

A central goal of the approach we promote in this paper is to cleanly separate a logical notion of service, which captures the capability that emerges from the interplay of a set of components, from its mapping to a concrete deployment architecture. In short, services are the centerpiece of logical architecture design, whereas components implement the services at runtime. We employ mechanisms of *model-based development* to accomplish this separation. Logical models describe capabilities in terms of functionality, distribution of components and quality properties independently of any implementation details or deployment architecture design decisions. The implementation models contain these details and decisions, and must be a consistent refinement of the logical models. This distinction has the advantage that many implementation models exist, which satisfy one logical model and that implementation models can be developed strictly after the logical models.

Technical complexities, such as defining the functional behavior of the system by identifying the services and their dependencies, designing objects/components and their interfaces so that they can provide the services with the required quality properties, and deploying them on a given middleware for most efficient operation are only part of the challenge. Organizational requirements within the project, the need for consistent documentation and efficient tool support, and extended system maintenance time spans impose further challenges. Development processes provide systematic support in some or all of these regards to various degrees.

The V-Modell XT [4, 30], for instance, is a modern software and systems development standard covering project management, engineering and supporting processes. It provides a generic process model, which is easy to understand and to use, and flexibly adaptable to the needs of organizations and projects. The V-Modell XT promises to lead reproducibly to project results of higher quality with less cost and resources spent. It is generic in nature and does not provide detailed methodic instructions for systems development; it has a modular setup enabling flexible extension.

Our goal is to provide an adapted, widely accepted development process that is flexible enough to support our service-oriented development models through all stages of system development and maintenance, without restriction to certain application domains. This will increase the applicability of our approach and help to gain wider acceptance beyond the research community. In this paper, we will present an integration of our service-oriented development approach with the V-Modell XT, which fulfills the requirements we have presented above and can be tailored to a wide range of development processes for ULS systems.

Service-Oriented Specifications The development methodology we propose in this paper focuses on services as first class entities throughout the development process of ULS systems; it establishes a clean separation between the services provided by the system under consideration, and the architecture – comprised of components and their relationships – implementing the services. Our approach is well suited for process embedding and tool support.

Following [14, 20] we use the notion of *service* to decouple abstract behavior from implementation architectures supporting it. Typically, services coordinate workflows among domain objects; they may also call, and thus depend on, other services. In this sense, services are specializations of use cases to specify interaction scenarios; services "orchestrate" the interaction among certain entities of the system under consideration to achieve a certain goal [6]. In contrast to use cases, which describe functionality typically in prose and on a coarse level of detail, a service is defined via the interaction pattern among a set of collaborators required to deliver the functionality. Services are partial interaction specifications.

As methodological core ignoring any management and QA aspects for now, we employ a two-phase, iterative development process (see [14]). In the first phase, services are elicited from use cases, and captured in terms of interaction patterns. In the second phase, a deployment architecture is defined as a set of interacting components; then the services are mapped to the deployment architecture to yield the overall software and systems architecture.

The relevant use cases and their relationships are defined as use case graph. This yields a relatively large-scale, scenario-based view on the system. From the use cases, sets of *roles* (actors) and *services* (functions) as interaction patterns of roles are derived. Using roles decouples from interaction details, because roles abstract from components or objects. Roles describe the contribution of an entity to a particular service independently of what concrete implementation component will deliver this contribution. An object or component of the implementation typically will play multiple roles at the same time. The communication relationships between roles are captured in a *role domain model*. The set of services is *mapped* to a *component configuration* refining the *role domain model* to yield an *architectural configuration*. These architectural configurations can be readily implemented, for instance prototypically by code generation.

Contributions and Outline As main contribution, this work presents an integration of a systematic approach for the development of ULS systems into an established and widely accepted generic process standard. We explain our modeling methodology and present its application by means of a running example through all stages of development. We show how we integrate this approach into the existing large-scale development model, V-Modell XT.

In Sect. 2, we introduce a service-oriented model of our running example, the Bay Area Rapid Transit system (BART). In Sect. 3, we explain the basic concepts and extension mechanisms of the V-Modell XT, which we integrate with our service-oriented modeling approach in Sect. 4. In Sect. 5, we discuss advantages and shortcomings of this integration. Sect. 6 contains related work and Sect. 7 presents conclusions and an outlook.

2. Service-Oriented Development of Complex Systems

The systems we address with our service-oriented approach are complex distributed systems. The complexity we refer to here stems from the need to integrate multiple different parts whose interplay is difficult to grasp with traditional techniques. Rather than treating component interplay as an afterthought, addressed only during late stages of deployment and integration, we focus on *services*, defined as the interaction patterns among roles, throughout the development process.

The BART Case Study The BART case study [41] describes parts of the Advanced Automatic Train Control (AATC) of the Bay Area Rapid Transit (BART) system. BART is the San Francisco area, heavy commuter rail train system. The case study describes the part of the train system that controls speed and acceleration of the trains. BART was previously used as a case study in the area of distributed systems and for the application of formal methods [11].

The BART system automatically controls over 50 trains on a large track network. Manual operation of the train control is limited mostly to safety issues and to cases of emergency or malfunction. The AATC system controls the train movement with a requirement to optimize train throughput, while constantly ensuring safety. The AATC system operates computers at the train stations which each control a local part of the track network. Stations communicate with the trains via a radio network and with neighboring stations. Trains receive acceleration and brake commands from the station computers and feed back speed, position and status information.

We focus on modeling the reactive behavior of a station computer and of the trains. We apply our service-oriented development approach to identify the different services of the system and to specify a service model that will help us to design an effective service-oriented architecture. Our model-based approach results in a high level design model of the AATC that can be refined systematically into an implementable system. In the following, we will explain the steps in more detail (see also [15]).

Use Case Elicitation From the requirements [41], we identify a number of use cases, for instance, "A *train* communicates its current status to the responsible *control station*". Each use case can be broken down into more detailed steps, leading to a comprehensive use case view of the BART system. Analyzing the use cases leads to an initial list of actors, or *roles*; in our case these are Train, Control Station, the Safety Computer (VSC) and an External Data Source as actors. We depict roles and their communication relations in an initial role domain model, as shown in Fig. 1.

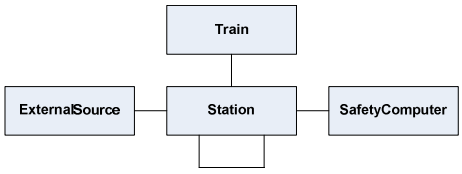

Figure 1. High Level BART Roles

Modeling Services and Roles We model roles and services, starting with the above initial role domain model, by systematically going though the list of use cases and identifying interaction patterns that define services. The services we identify are somehow a refinement and formalization of the use cases. In the process of identifying interaction patterns, we may identify further actors; we add these as roles to the role domain model. Finally, after modeling all services the resulting role domain model looks as depicted in Fig. 2.

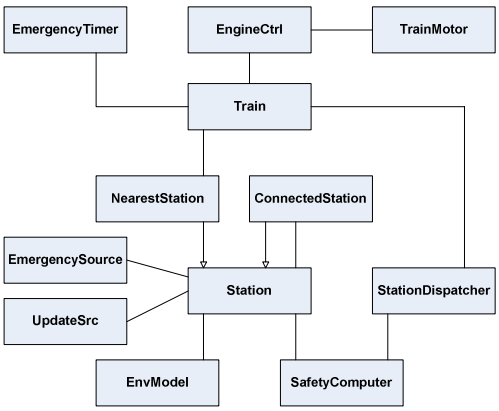

Figure 2. BART Role Domain Model

We use the extended MSC notation of [13, 15] to specify services. This notation is based on the Message Sequence Chart [9] standard and provides an intuitive graphical language for specifying interaction patterns and is well-accepted among engineers. Extensions to the standard notation were cautiously made based on a formal semantics to provide increased expressiveness and more powerful operators suitable for modeling service-oriented systems. To model the services, we make use of our tool-chain introduced in [1, 5].

We capture the interactions between the station computer (and its subcomponents) with a train (and its subcomponents). Other entities, such as external data sources, are part of the interactions as well. When modeling the interactions, we abstract from any concrete deployment architectures and do not consider multiple occurrence of the same entities. For instance, we specify the interactions between a train and the station computer abstractly, not knowing how often this interaction will occur in the implementation.

Good design principles suggest a hierarchical design of the service model. We use High Level MSC (HMSC) to introduce hierarchy. Intuitively, an HMSC is a graph depicting a *flow* through a set of services. The HMSC "TrainLoop" in Fig. 3 shows an infinite flow of activities of regular train operation, potentially preempted by exceptional behavior in case of an emergency situation; then the emergency is resolved after which the behavior returns to normal operations.

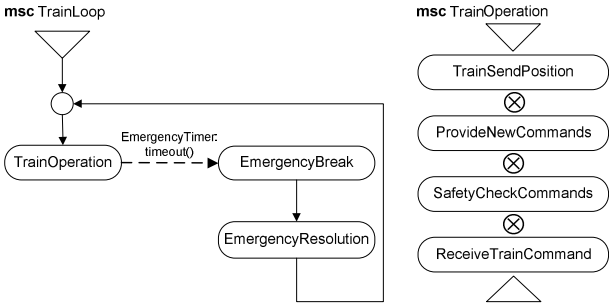

Figure 3. HMSCs for BART services

The "TrainLoop" HMSC contains MSC references, depicted by labeled rounded boxes, pointing to more detailed interaction behavior. The referenced functionality for "TrainOperation", for instance, is specified in the MSC shown on the right side of the figure. It shows a composition of four services by means of the "join"-Operator, depicted as $\otimes$. The semantics attached to the join of two services is the interleaving of the two behavior specifications, synchronized on common messages. The *join* operator is a powerful means to combine and synchronize *overlapping* services – this ability to disentangle service specifications is central in our approach. We call services overlapping if they share at least two roles and at least one message between shared roles. Overlapping interactions will occur only once – synchronized – in the resulting behavior. For details about the join operator, see [13, 19, 14]. We can also apply operators for *Sequence* of interactions, *Non-deterministic choice*, *Parallel* interactions and for *Preemption*, defining exceptional behavior.

Fig. 4 shows the behavior specification of a train sending current status values to the nearest station that processes the information in the syntax of an extended Basic MSC. Messages are depicted as horizontal arrows between two roles (represented as vertical axes labeled with the name of the role) and have parameters to indicate transmission of data values.

We make use of MSC operators, depicted as labeled boxes, to express repetition and choice in the interaction flow. The $LOOP<*>$ box around all the interactions in the MSC expresses repetitive behavior. Alternative or optional behavior is expressed by *ALT* boxes. Different alternatives are separated by horizontal dashed lines through the box. To indicate optional behavior, we leave one of the alternatives empty. Further operators at our disposal are for *parallel*, *joined* and *preempted* interactions. With *TRIGGER*, we express liveness conditions. State markers (hexagons) define states in the execution of a role. For more information and precise semantics definitions, see [13].

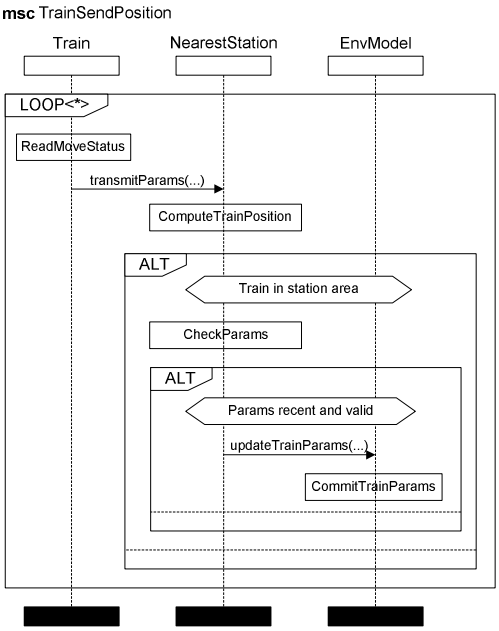

Figure 4. MSC TrainSendPosition

We integrate control aspects into reactive interaction specifications by means of *local actions*. Local actions are depicted as labeled boxes on role axes. The meaning of this syntax is that a role performs an activity based on the information available until this point in time. Information can be local variables, data previously received via messages and the role state.

Mapping the Service Model to Components When transitioning from a service model with roles and interactions to an implementable architecture, we define *component types*, which are blueprints of *component instances* in the architecture. We define their communication interfaces to other component types and the services they implement. The component model needs to be a refinement of the structural role dependencies. Table 1 shows an example role-to-component mapping. The behavior of the component types can be derived for instance from the service specifications using the component synthesis algorithm, described in [20].

Component Type	Role	Description
FastCPU	Station EnvModel StationDispatcher	A fast CPU computer for operative station control
SlowCPU	SafetyComputer	High Reliability (MTBF) slow CPU unit checking safety conditions
Train	Train TrainMotor EmergenyTimer EngineCtrl	Train computing unit on board of a train
Interlocking-System	UpdateSrc TrainMotor	Interlocking system, controlling switches and gates

Table 1. BART Role Mapping

Defining a Component Architecture Fig. 5 shows a simple architecture, which implements the service model that we have specified above. It shows the structure of the system's components and their connections. Components are instances of component types and can occur multiple times in a system configuration. Each instance has a defined name and specific type.

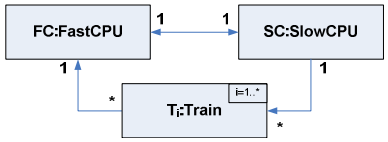

Figure 5. BART Component Architecture

3. The V-Modell XT

V-Modell XT Concepts The V-Modell XT [4, 30] is a modern German software and systems development standard; its application is mandatory for all contractors supplying IT systems to the German federal government and military. The V-Modell XT has a modular setup and enables flexible usage and extension. In order to extend the V-Modell XT it is imperative to know its main concepts, which we explain in the following:

- *Work Products* are the essential project results and artifacts (documents, models, code, deliverable systems). They have a prescribed structure and content and can be structured further into subjects (subsections). Work products have a responsible role and will be quality evaluated.
- *Product Dependencies* define consistency relations between the contents of different work products. This will keep new work products in a project consistent to existing ones, assuring overall product quality and information traceability.
- *Activities* define actions that need to be performed in order to edit work products. One activity is associated exactly with one work product. Activities can be structured further into sub-activities.
- *Roles* describe profiles of responsibility for individuals working in a project.

- *Process Modules* group Work Products, Activities, Roles, and other V-Modell XT elements into self-contained units with a common purpose such as project management, requirements management, systems development, etc. Process modules may depend on others. They can be understood, applied and modified independently and are the units of tailoring and extension of the V-Modell XT.
- *Tailoring* is the process of adapting the V-Modell XT to a specific project or organization, by selecting the suitable process modules out of the repository of available ones. Tailoring results in a seamless consistent adapted software development process.

Systems Development The V-Modell XT defines many processes which are important for the development of systems, including management processes (such as "Offer Preparation and Contract Fulfillment"), engineering processes (such as "Specification of Requirements"), as well as supporting processes (for instance "Quality Assurance"). Each of these processes is defined in a process module. In the following, we concentrate on the for main system development process modules:

- *Specification of Requirements*: The process of preparing a requirements specification document based on a project proposal and evaluating these requirements in terms of effort, cost and importance.
- *System Development*: The process of decomposing a complex system into manageable units of software, hardware, supporting systems and additional materials (such as manuals) and of integrating the units to the deliverable system.
- *Software Development*: The process of developing an individual unit of software, which includes specification, software architecture design, implementation and integration, test specification and unit evaluation.
- *Hardware Development*: The process of developing an individual unit of hardware.

These four process modules define work products (artifacts), activities and roles that are required to develop a system from customer defined requirements specification to the acceptance of the final deliverable. Fig. 6 depicts the different stages of development in the shape of the "V", which is often associated with the V-Modell XT. System development involves the creation of a number of significant project results with clearly defined role responsibilities. Activities describe how the project results are created.

Requirements are collected in the *Requirements Specification* by the *Requirements Engineer (Acquirer)*. The document contains a situational overview, a list of functional and non-functional requirements, the list of deliverables with

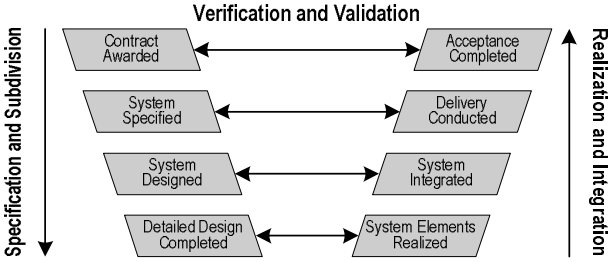

Figure 6. V-Modell XT Development Stages

acceptance criteria and other subjects. Functional requirements are defined as use cases. The *Requirements Engineer (Supplier)* is responsible for refining the Requirements Specification into an *Overall System Specification*, which is the basis for any system development and documentation activities. The overall system level comprises Integrated Logistic Support (ILS) documentation, enabling systems and the technical system under development, and contains an overall system architecture and the list of system interfaces. We concentrate in the following on the technical system.

The V-Modell XT divides a technical system into system and unit level. The system contains hardware, software and documentation parts which can be developed, acquired or pre-existing. Systems decompose into segments (if necessary) and at the lowest level into units. Units are either pure software, hardware or external. Software/hardware units decompose further into components (if necessary) and modules. The V-Modell XT has a uniform structure for specifying and designing these system elements:

- A *Specification* document describes the context and purpose of a system element and its interfaces from a black-box view, as well as non-functional requirements and internal interfaces between sub-elements.
- An *Architecture* document describes the architecture of one system element and its parts, by documenting architectural principles, design alternatives, decomposition into sub-elements, cross-cutting concerns (such as transaction and security handling), internal interfaces and dependencies, and more.
- An *Implementation, Integration and Evaluation Concept* describes details and plans about the actual implementation with tools and procedures used, the integration and the execution environment, as well as any test and verification strategies.

The uniform structure of system element specification, architecture design and decomposition makes development very systematic, despite the substantially different tasks of system, software, and hardware engineering. Product dependencies ensure that the contents in all products are

consistent, for instance that all requirements are realized by architecture elements. Role responsibilities depend on the level of decomposition: *System Architect*, *Software Architect*, *Hardware Architect*, *Software/Hardware Developer* and *System Integrator* are roles that create the respective products, by performing the activities (with sub activities) associated with the products.

V-Modell XT Extension Mechanism The V-Modell XT can be modularly extended by adding new process modules, which extend a number of existing process modules. A new process module can contain definitions for new work products, activities and roles. New product dependencies to existing products will make sure that the new products are created when required. Seamless extension is possible by adding new subjects, and sub-activities to existing elements.

4. Extending the V-Modell XT

As described above, the V-Modell XT provides a full workflow and process for systems development. It does not, however, give specific methodological guidelines for service-oriented development. We will provide this in the following by describing the integration of our service-oriented development approach into the *V-Modell XT* [4].

We define a new process module "Service-Oriented Systems Development" as shown in Fig. 7. We make use of the fine-grained extension mechanism by adding new subjects to existing work products and new sub-activities to existing activities. Additionally, we add product dependencies to keep our additions consistent with the rest.

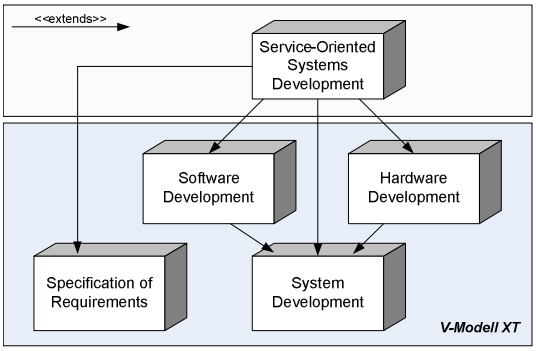

Figure 7. Service-Oriented V-Modell XT Extension

Work products are the main V-Modell XT elements and core project results. Table 2 shows a mapping of the result artifacts of our service-oriented approach to work products and their subjects of the V-Modell XT. V-Modell elements marked with an asterisk (*) are additions. We follow the V-Modell's hierarchical system decomposition into system elements similarly for our service-oriented approach. We define a service model and mapping to an architecture decomposition for each system element with specification and architecture documents. Our service-oriented approach thus scales in the same way as the V-Modell XT.

Element	V-Modell Element	Description
Use Case Graph	Requirements Spec.: *Functional Req's* Overall System Spec: *Functional Req's*	Use Case descriptions are part of the existing requirements documentation.
Services	System/SW/HW Spec.: *Service Access Points** System/SW/HW Arch.: *Service Specifications**	Service interfaces, accessible from outside; service specifications as interaction patterns.
Roles	System/SW/HW Arch.: *Role Model**	The list of roles, their descriptions and the states they can be in.
Role Domain Model	System/SW/HW Arch.: *Role Model**	Description of communication channels between roles.
Compon. Configuration	System/SW/HW Arch.: *Decomposition*	The sub-element structure as known.
Mapping	System/SW/HW Arch.: *Role to Component Mapping**	Maps roles (and thereby service behavior) to sub-elements.
Architecture	System/SW/HW Impl., Int. and Eval. Concept: *Integration Procedures*	The exact deployment configuration as instance network.

Table 2. Product Mapping

We add the following product dependencies that ensure consistency across work products:

- *Consistent Service Refinement*, between products *Overall System Specification* and *Architecture*: the services and roles realize the defined use cases and non-functional requirements.
- *Consistent Service Model*, between products *Specification* and *Architecture*: All roles used within service specifications must be defined; the role structure must reflect the interaction patterns from the service specifications; the service access points must match the service specifications.
- *Consistent Service Mapping*, between products *Specification* and *Architecture*: the sub-element structure as given by the decomposition must be a refinement of the role model; all roles and services must be mapped to sub-elements.

We do not require additional activities; instead we extend existing ones with relevant sub-activities. The sub-activity definitions follow our explanations in Sect. 2:

- The V-Modell activities *Preparing System/Software/Hardware Specification* get the new sub-activities *Defining Service Interfaces*.
- The V-Modell activities *Preparing System/Software/Hardware Architecture* get the new sub-activities *Defining Roles and Role Structure*, *Defining System Element Services*, and *Mapping Roles to System Elements*

The integration of our model affects the responsibilities of five existing roles of the V-Modell XT. Their role profile descriptions are sufficiently abstract and fit our purposes. Thus, only slight extensions need to be performed:

- For the roles *Requirements Engineer (Acquirer)* and *Requirements Engineer (Supplier)*, we add domain-modeling and service-oriented design as required capability.
- For the roles *System Architect*, *Software Architect* and *Hardware Architect*, we require knowledge of service-oriented design and of service-oriented architectures and infrastructures.

5. Experiences and Discussion

We have applied the extended V-Modell XT to our BART example in the following way: We start with the *Requirements Specification* containing the system use cases extracted from [41]. We begin system development by refining the requirements into the *Overall System Specification* document, where we specify the BART system as our technical system. The *System Architect* designs the initial *System Architecture* similar to Fig. 1, consisting of the segments Train, Station, ExternalSource, and SafetyComputer, taking the architectural constraints of [41] into account. We analyze the use cases and identify the roles and services as explained in Sect. 2. We list the roles and their connections (see Fig. 2) in the *Role Model* subject and the services (as specified in Fig. 3 and 4) in the *Service Specification* subject of the *System Architecture*. We define the external service access points in the *System Specification*, for instance the communication protocol between Train and Station. With the domain knowledge from the service elicitation steps, the *System Architect* designs the actual *System Decomposition* in the *System Architecture* containing the elements given in Fig. 5. We follow the procedure of Sect. 2 and document the mapping of roles to services in the subject *Role to Component Mapping* of the *Software Architecture*, as shown in Table 1. The actual deployment architecture (see also Fig. 5) is described in the *System Impl., Integration and Eval. Concept*. The system decomposes further into software, hardware and external units (such as the Interlocking System), and into components and modules. We do not describe the detail level specifications and architectures; they follow the standard V-Modell XT development process without service modeling. We perform V-Modell XT activities to edit the work products. Each work product is subject to a Quality Assurance procedure, which includes checks of the product dependencies, including the ones listed above.

Following the above, we have conducted the BART case study as service-oriented V-Modell XT development project. We have shown how the artifacts of our service modeling approach fit in the V-Modell XT work results, in general and for our case example. In summary, we find that our approach blends well with the V-Modell XT. The necessary changes and additions to use the model fit with the existing structure and require only additions and slight extensions of existing V-Modell XT elements. They can all be packaged nicely as a process module. An embedding into other process models with a similar structuring, such as the RUP [12], should be possible with a comparable effort.

One of the challenges of integrating our model-based service-oriented development approach into the V-Modell XT is the V-Modell's strict notion of work products as sole project results. Work products are mostly documents including design model views that describe the system under development. Model-based approaches on the other hand often make use of integrated models, modeling environments and highly iterative modeling cycles. With good tool support, it is possible to keep the service-oriented model consistent across a number of documents, hierarchies and iterations.

Another challenge lies in the structure of the V-Modell XT system elements. The technical system (as part of the overall system) is hierarchically decomposed into segments and these into units, which are either software or hardware. Units have their own specifications and architecture documents and decompose into components and modules. Services in our understanding are cross-cutting across all structural system elements. They are defined as interaction patterns (or protocols) between interacting components. The solution lies in the strategy how the *System Architect* decomposes a system compliant to the V-Modell. The system elements need to be defined such that services can span all required components. New levels of hierarchy are suitable, where services cross-cut only direct subcomponents. This creates a layered service (and system) hierarchy.

6. Related Work

The notion of service and service-oriented development is used in many different application domains and on various levels of abstraction in the Software Engineering community [36, 18]. Its roots lie in the domain of telecommunication systems, where features and their interactions play an important role in software development [37, 28, 29, 42].

Intensive application of service-oriented approaches can be observed for web services [26, 39], web service-oriented architectures [32, 39] and increasingly for embedded automotive systems [21, 17, 3]. Implementation-oriented and infrastructure concerns, including web services [32] and corresponding technologies, such as WSDL [38], WS-BPEL[22], .NET [27] and J2EE [34], or specific mechanisms such as registration, discovery and binding are relevant as members of the deployment technology space for service-oriented development. In the realm of web services standards there has been important work on Web Services Semantics (WSDL-S) [40], Web Services Modeling Ontology (WSMO) [31] and Semantic Markup for Web Services (OWL-S) [25]. OWL-S, in particular, describes both a "service profile" and a "service grounding", which represent "what the service does" and how it maps to underlying messaging protocols and deployment technologies, respectively. To describe the functionality of a service, OWL-S uses a process notion, which uses a limited set of operators to build composite processes – in particular, the notion of joining overlapping services is missing from this approach. The deployment model underlying all of OWL-S and WSDL-S is oriented more towards services as I/O processes, whereas our service notion is closer to the "conversations" or orchestration underlying WS-BPEL [22]; WS-BPEL, however, also would benefit from an operator for disentangling services, such as the join operator we have introduced, above.

Because our approach rests on an end-to-end, interaction-pattern-based service definition (see below), it integrates well with ontologies [25, 31] capturing non-functional aspects – including, but not limited to, security and Quality-of-Service [1].

Our approach is related to the Model-Driven Architecture (MDA) [23], Model-Integrated Computing [35, 10], aspect-oriented modeling (AOM) [7] and architecture-centric software development (ACD) [24]; similar to MDA and ACD we also separate the software architecture into abstract and concrete models. In essence, the service elicitation and architecture definition phases in our development process correspond to building "platform-independent models" (PIMs) and "platform dependent models" (PDMs), respectively. In contrast to MDA and ACD, however, we consider services and their defining interaction patterns as first-class, cross-cutting modeling elements of both the abstract and the concrete models. This also distinguishes our approach from Model-Integrated Computing and AOM.

Ambler uses process patterns in [2] to describe task-specific self-contained pieces of processes and workflows in a reusable way. Such patterns can be applied to solve complex tasks when needed. Störrle [33] shows how process patterns can be described in great detail using UML. The idea of process patterns is further refined by Gnatz et al. [8] in the form of a modular and extensible software development process based on collections of independent process components. These process patterns essentially are the basis of the extension mechanism of the V-Modell XT.

7. Summary and Outlook

Service-oriented development promises to address many complexities in the development of ultra-large scale (ULS) systems. We have explained our service-oriented approach and corresponding notations using the BART system case study. Our approach seems to be well suited to managing the complexity of this distributed, reactive system, by focusing on services as first-class entities throughout the development process.

We have integrated our service-oriented development approach into an existing systems development process model by describing a service-oriented extension of the V-Modell XT. We have shown how our approach can be realized through extensions of V-Modell activities and product definitions. We ensure consistency to existing V-Modell concepts by introducing product dependencies. The V-Modell XT extension appears to be seamless and intuitive. We have enriched the flexible organization and management framework with valuable methodical detail for service-oriented development. Thus, we support the development of ULS systems by suitable models and notations as well as by a large-scale systematic development process.

Future work will include conducting a larger case study or V-Modell XT pilot project with the service-oriented extensions. We would also like to refine our extensions to include non-functional requirements and quality properties.

References

[1] J. Ahluwalia, I. Krüger, M. Meisinger, and W. Phillips. Model-Based Run-Time Monitoring of End-to-End Deadlines. In *Proceedings of the Conference on Embedded Systems Software (EMSOFT)*, 2005.

[2] S. W. Ambler. *More Process Patterns: Delivering Large-Scale Systems Using Object Technology*. Cambridge University Press, 1999.

[3] M. Broy, I. H. Krüger, and M. Meisinger, editors. *Automotive Software - Connected Services in Mobile Networks. Proceedings of the Automotive Software Workshop San Diego 2004*. Lecture Notes in Computer Science, Volume 4147, Springer, New York, 2006.

[4] Bundesrepublik Deutschland. *V-Modell XT*, 1.2 edition, 2006. `http://www.v-modell-xt.de/`.

[5] V. Ermagan, T.-J. Huang, I. Krüger, M. Meisinger, M. Menarini, and P. Moorthy. Towards Tool Support for Service-Oriented Development of Embedded Automotive Systems. In *Proceedings of the Dagstuhl Workshop on Model-Based Development of Embedded Systems (MBEES'07)*, To appear as LNCS. Springer Verlag, 2007.

[6] E. Evans. *Domain Driven Design*. Addison-Wesley, 2003.

[7] G. Georg, R. France, and I. Ray. Composing Aspect Models. In *Proceedings of the 4th AOSD Modeling With UML Workshop*, 2003.

[8] M. Gnatz, F. Marschall, G. Popp, A. Rausch, and W. Schwerin. The living software development process. *Software Quality Professional*, 5(3), June 2003.

[9] ITU-TS. Recommendation Z.120: Message Sequence Chart (MSC). Geneva, 1996.

[10] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-Integrated Development of Embedded Software. In *Proceedings of IEEE January 2003*, 2003.

[11] F. Kordon and L. Michel, editors. *Formal Methods for Embedded Distributed Systems*. Springer, 2004.

[12] P. Kruchten. *The Rational Unified Process. An Introduction*. Addison-Wesley, 2nd edition, 2000.

[13] I. Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.

[14] I. Krüger, R. Mathew, and M. Meisinger. Efficient Exploration of Service-Oriented Architectures using Aspects. In *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, 2006.

[15] I. Krüger, M. Meisinger, and M. Menarini. Applying Service-Oriented Development to a Complex System: the BART case study. In *Proceedings of the Monterey Workshop 2005*, number 4322 in LNCS. Springer Verlag, 2007.

[16] I. Krüger, M. Meisinger, M. Menarini, and S. Pasco. Rapid Systems of Systems Integration - Combining an Architecture-Centric Approach with Enterprise Service Bus Infrastructure. In *Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 51–56. IEEE, 2006.

[17] I. Krüger, E. C. Nelson, and V. Prasad. Service-based Software Development for Automotive Applications. In *CONVERGENCE 2004*, 2004.

[18] I. Krüger, B. Schätz, M. Broy, and H. Hussmann, editors. *SBSE'03. Service-Based Software Engineering. Proceedings of the FM2003 Workshop*, number TUM-I0315, 2003.

[19] I. H. Krüger. Capturing Overlapping, Triggered, and Preemptive Collaborations Using MSCs. In M. Pezzè, editor, *FASE 2003*, volume 2621 of *LNCS*, pages 387–402. Springer Verlag, 2003.

[20] I. H. Krüger, R. Mathew, S. Leue, and T. Systä. Component Synthesis from Service Specifications. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 255–277. Springer Verlag, 2005.

[21] E. C. Nelson and K. V. Prasaad. Automotive Infotronics: An emerging domain for Service-Based Architecture. In I. H. Krüger, B. Schätz, M. Broy, and H. Hussmann, editors, *SBSE'03 Service-Based Software Engineering, Proceedings of the FM2003 Workshop*, Technical Report TUM-I0315, pages 3–14. Technische Universität München, 2003.

[22] OASIS. Web Services Business Process Execution Language (WS-BPEL), Version 2.0. Specification public draft, 23-Aug-2006, 2006. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf`.

[23] OMG (Object Management Group). Model Driven Architecture (MDA). MDA Guide 1.0.1, omg/03-06-01, 2003. `http://www.omg.org/mda`.

[24] OMG (Object Management Group). UML, Version 2.0. OMG Specification formal/05-07-04 (superstructure) and formal/05-07-05 (infrastructure), 2005.

[25] W3C: OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004, 2004. `http://www.w3.org/Submission/OWL-S/`.

[26] C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, 2003.

[27] D. S. Platt and K. Ballinger. *Introducing Microsoft .NET*. Microsoft Press, 2001.

[28] C. Prehofer. Feature Oriented Programming: A fresh look at objects. In *Proceedings of ECOOP 1997*, number 1241 in LNCS. Springer Verlag, 1997.

[29] C. Prehofer. Plug-and-Play Composition of Features and Feature Interactions with State-chart Diagrams. In *Proceedings of the 7th International Workshop on Feature Interactions in Telecommunications and Software Systems*, 2003.

[30] A. Rausch and M. Broy. *Das V-Modell XT. Grundlagen, Erfahrungen und Werkzeuge*. dpunkt.verlag, 2006. In German.

[31] D. Roman, U. Keller, H. Lausen, R. L. J. de Bruijn, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.

[32] J. Snell, D. Tidwell, and P. Kulchenko. *Programming Web Services with SOAP*. O'Reilly, 2002.

[33] H. Störrle. Describing Process Patterns with UML. In *Proc. of the 8th European Workshop on Software Process Technology (EWSPT '01)*, pages 173–182. Springer, 2001.

[34] SUN Microsystems Inc. Java Platform, Enterprise Edition (Java EE, J2EE). Website, 15-Nov 2006. `http://java.sun.com/javaee/`.

[35] J. Sztipanovits and G. Karsai. Model-Integrated Computing. *IEEE Computer*, 30(4):110–112, April 1997.

[36] D. Trowbridge, U. Roxburgh, G. Hohpe, D. Manolescu, and E. Nadhan. *Integration Patterns. Patterns & Practices*. Microsoft Press, 2004.

[37] K. J. Turner. Relating Services and Features in the Intelligent Network. In *Proc. of the 4th International Conference on Telecommunications*, pages 235–243, 1997.

[38] W3C. Web Services Description Language (WSDL), Version 1.1. W3C Note, 15-Mar-2001, 2001. `http://www.w3.org/TR/wsdl/`.

[39] W3C. Web Services Architecture, 11-Feb 2004. `http://www.w3.org/TR/ws-arch/`.

[40] W3C: Web Service Semantics - WSDL-S, Version 1.0. W3C Member Submission, 7-Nov-2005, 2005. `http://www.w3.org/Submission/WSDL-S/`.

[41] V. Winter, F. Kordon, and L. Michel. The BART Case Study. In F. Kordon and L. Michel, editors, *Formal Methods for Embedded Distributed Systems*, pages 3–22. Springer, 2004.

[42] P. Zave. Feature-Oriented Description, Formal Methods, and DFC. In *Proceedings of the FIREworks Workshop on Language Constructs for Describing Features*, pages 11–26. Springer-Verlag, 2001.