Template-Based Adaptation of Semantic Web Services with Model-Driven Engineering

Athanasios Staikopoulos, Owen Cliffe, Razvan Popescu, Julian Padget, and Siobhán Clarke

Abstract—Service-oriented enterprise systems, which tend to be heterogeneous, loosely coupled, long-lived, and continuously running, have to cope with frequent changes to their requirements and the environment. In order to address such changes, applications need to be inherently flexible and adaptive, supported by appropriate infrastructures. In this paper, we propose a model-driven approach for the dynamic adaptation of Web services based on ontology-aware *service templates*. Model-driven engineering raises the level of abstraction from concrete Web service implementations to high-level service models, which leads to more flexible and automated adaptations through template designs and transformations. The ontological semantics enhances the service matching capabilities required by the dynamic adaptation process. Service templates are based on OWL-S descriptions and provide the necessary means to capture and parameterize specific behavior patterns of service models. In this paper, we apply our approach in the context of the EU-funded ALIVE project and illustrate, as an example, how the proposed framework supports the adaptation of the authentication mechanism used by an interactive tourist recommendation system.

Index Terms—Evolving Internet applications, methodologies, software architectures, support for adaptation, Web services, Web services modeling, formalization of services composition, process re-engineering methodology.

1 Introduction

The reality of many of today's software systems is that their operating environments are highly dynamic. Changes to the environment and indeed, to their requirements, happen continuously, both explicitly and implicitly. This is more and more evident for systems that tend to be heterogeneous, loosely coupled, long-lived, and that are required to run continuously, such as business-based, service-oriented systems.

Web services provide the basic fundamental unit for constructing such systems, representing a specific business activity or functionality. Once services are exposed and become available for use by other services and resources on the Web, they can establish more complicated structures and interactions, providing new added value aggregate services. In order to effectively support the highly dynamic nature of such systems, services need to be handled as flexible, composite, and adaptive units, so they can be substituted, converted, or even composed easily with other services.

In general, adaptation is an essential quality for services that operate within dynamic environments and provide high availability with reduced system downtime. Through adaptation, service-oriented systems can achieve higher levels of maintenance and autonomy. The adaptation process serves, for example, to ensure the following:

- The application is compatible with all its clients (e.g., an e-commerce platform has to be compatible with both old and new clients).
- The application maintains a desired quality of service (QoS) (e.g., an online video-sharing application has to scale up with an increasing number of active clients).
- The application is fault tolerant (e.g., an application should provide an alternative authentication service, should the main one fail).

An adaptation can be enforced either at design or at runtime, and it can be triggered either by the human designer or operator of the application, or by a monitoring process. Several techniques have been defined for the monitoring and adaptation of applications. They generally tackle issues such as interface [1], behavioral [2], quality of service [3], service-level agreement [4], or policy mismatches [5]. However, such techniques usually work in isolation and cannot be easily integrated to tackle complex monitoring and adaptation scenarios [6].

A consolidated and flexible framework is required to allow developers to integrate such adaptation techniques to tackle complex application requirements within such dynamic environments. The framework should monitor the execution of services within their operating environment and should direct and trigger dynamic adaptations on services, once problems or requirements arise.

In addition, a methodology that supports the effective design, development, and native capabilities of such systems is essential [7]. Model-driven engineering (MDE) [8] offers a number of benefits to the software development process by raising the level of abstraction in which we develop software from low-level concrete implementations to high-level model abstractions. So, it is possible to reason about the properties of a system through visual representations (models) and

- *A. Staikopoulos, R. Popescu, and S. Clarke are with the School of Computer Science and Statistics, Trinity College Dublin, University of Dublin, Dublin 2, Ireland.*
  E-mail: {athanasios.staikopoulos, rpopescu, siobhan.clarke}@cs.tcd.ie.
- *O. Cliffe and J. Padget are with the Department of Computer Science, University of Bath, Bath BA2 7AY, UK.*
  E-mail: o.c.cliffe@bath.ac.uk, jap@cs.bath.ac.uk.

Manuscript received 30 Apr. 2009; revised 29 Oct. 2009; accepted 3 Jan 2010; published online 26 May 2010.
For information on obtaining reprints of this article, please send e-mail to: tsc@computer.org, and reference IEEECS Log Number TSCSI-2009-04-0095.
Digital Object Identifier no. 10.1109/TSC.2010.30.

1939-1374/10/$26.00 © 2010 IEEE Published by the IEEE Computer Society

automatically create implementations for a variety of problems via transformations. MDE also provides all the necessary ingredients (e.g., theoretical foundations, standards, and toolsets) for the systematic engineering, modeling, and automation of dynamic service-oriented systems and their adaptations.

The ALIVE project [9], [10] proposes an advanced MDE-supported framework for the disciplined, systematic, and engineered development and management of service-oriented systems, based on coordination and organization mechanisms often seen in human and other societies.

In this paper, we describe an approach for the dynamic adaptation of services supported by the ALIVE framework. The adaptation approach consists of an MDE process based on ontology-aware service templates. This process leads to flexible adaptations through design templates and automated transformations. The ontological semantics enhances the service matching capabilities required by the dynamic adaptation process. Service templates are based on OWL-S descriptions [11] and provide the necessary means to capture and parameterize specific behavioral patterns of service models, for example, capturing a common "authentication" mechanism.

In the context of this paper, service adaptations are considered as substitutions (for runtime adaptation), conversions, composition, or direct modifications of the service properties and parts (for design-time adaptation). A given service is substituted by an equivalent one or synthesized (composed) from existing services. The synthesis process is facilitated by the specification of service templates at design time. Developers may generate templates either manually or semiautomatically with the help of third-party adaptation techniques [1], [2], [3], [4], [5]. Service templates capture well-known types and patterns of service interactions, providing adaptation solutions for a specific type of problem. They are used for applying planned adaptations according to a given service type. Alternatively, runtime adaptations are supported via the direct substitution of services from equivalent ones via semantic matching and configuration.

In a nutshell, our MDE-based adaptation approach brings the following advantages:

- It allows the capture of high-level abstractions of the domain through service models and templates (patterns). The same service model or template can be synthesized into concrete services and adapters, in different applications.
- It sets the basis for the integration of third-party adaptation techniques needed to tackle complex application requirements (e.g., employ existing adaptation techniques for the generation of service templates).
- It enhances the generation of services and adapters from service templates from a manual, error-prone process to a semiautomated, engineered one.
- It allows for the development of heterogeneous services and adapters (through the use of service models and templates as common service and adapter description language).
- It supports the development of tools that assist the developer in the process of generating and customizing services and adapters.

The flexibility introduced by the MDE approach and the use of service templates allows application developers to apply the adaptation process to various application domains, such as e-commerce (e.g., to cope with changes in the business process requirements), crisis management (e.g., due to changes in the environment), and user entertainment (e.g., so as to offer customized services).

In this paper, we illustrate the applicability of our approach by showing how it can be employed for the runtime adaptation of an interactive tourist recommendation system. In particular, a detected failure on a security requirement (via a monitoring mechanism) triggers an adaptation process to replace the initial user authentication protocol with an alternative one. Similarly, other QoS properties such as performance and availability can be monitored and once problems are diagnosed, different adaptation templates that are in-place may be applied. These may replace communication protocols with more efficient ones or substitute services with more reliable ones.

The remainder of the paper is organized as follows: Section 2 provides background information. Section 3 presents a motivating example. Section 4 proposes a methodology for the adaptation of services based on a model-driven approach. Section 5 presents our approach in the context of the ALIVE framework. Section 6 provides a critical discussion of our approach. Section 7 describes related work. Finally, Section 8 summarizes the main contributions of our approach.

2 Background Information

In this section, we outline the basic concepts of the paradigms, technologies, and methodologies referred to across this paper.

2.1 Service-Oriented Computing and Architecture

Service-Oriented Architecture (SOA) is a standardized representation of the service-oriented computing paradigm, which defines a conceptual infrastructure for designing and developing service architectures, based on a set of open standards. SOA proposes a layered architecture for organizing services, which can be published, discovered, invoked, and combined to create more complex services [12]. Key SOA roles are the service provider, requestor, and registry.

2.2 Semantic Web Services

Web service descriptions (e.g., WSDL [13]) are superficial and lacking in any perspective of the service's semantics. As a result, searching, extracting, and matching services are correspondingly difficult, and limited by the lack of precision and depth in the description of each service. For example, UDDI [14] service registries feature keyword-based service matching mechanisms for WSDL services.

The semantic Web has emerged as a solution [15] extending the current Web technologies with well-defined meanings to existing services and resources. This is supported by annotations with (ontological) semantics through languages that can be interpreted and processed by computers. Ontologies [16] are used to provide a formal and explicit specification of the domain concepts, logical relations, restrictions, and properties used. Semantic reasoners use this information to perform automatic analysis and assertions of Web services and resources.

Consequently, semantic Web technologies provide additional scope for (semi-)automated service analysis, selection, and matching, providing automated processing and decision making based on semantic descriptions. These processes can then be exploited in the context of service composition and adaptation.

2.3 Semantics-Driven Service Matchmaking

Given a client query (e.g., consisting of desired service inputs and outputs), service matchmakers typically start by building a *candidate set* of services by querying available service directories for potentially matching services. For instance, a typical criterion may be whether service descriptions refer to similar terms or ontologies to those referred to by the query. Following the construction of a candidate set, for each service in the candidate set, a comparison is made against the query and a rank is computed based on one or more similarity metrics (e.g., [17]). Services may then be selected either automatically or manually according to their rank as part of a system (re-)configuration.

2.4 Service Adaptation

The unavailability of services, unexpected failures, changes of QoS requirements, alterations to communication protocols, and incompatibilities of the data exchanged can occur both explicitly and implicitly. These problems can be resolved to some degree through an adaptation process, where services are modified, substituted, or even composed transparently to perform their originally required or equivalent functionality. In this manner, adaptation can contribute to the high availability of services within changing environments. This is especially important for service-oriented business-based systems that are long-lived and continuously running and that operate within changing environments and requirements.

2.5 Model-Driven Software Engineering

A model-driven approach to development is generally based on a set of open standards and related technologies, and is built on a metamodel foundation, enabling a development framework for standard specification and interoperability among tools. Systems and applications are formalized with metamodel descriptions and are visualized by models as metamodel instantiations, using tools. Actual code implementations or other artifacts (e.g., jar) are created automatically by applying predefined transformations from source models to target models or languages.

Fig. 1 illustrates the model-driven process. Within the metamodeling layered foundation [18], at the top level, there is a meta-metamodel specifying the necessary constructs to build metamodels, such as for SOA. The metamodels themselves can be specified at varying levels of abstraction: from highly abstracted and independent models—Platform Independent Models (PIMs) to more technological and implementation specific—Platform-Specific Models (PSMs). Once a mapping is specified between corresponding metamodeling constructs, a transformation language implements the mapping and converts a model(s) of the source metamodel into a corresponding model(s) (M2M) or converts code (M2T) into another metamodel or language.

MDE offers a number of benefits to the software development process. In particular, it has the following benefits:

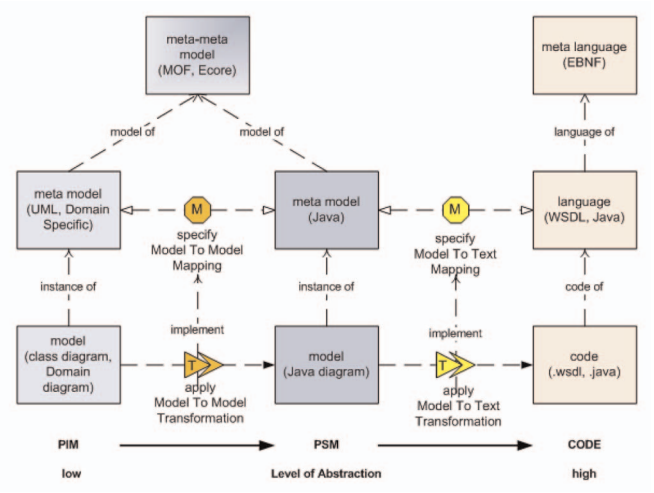

Fig. 1. Conceptualization of the model-driven approach.

1. It provides visual representations as an aid to communication and understanding.
2. It captures applications and systems at various levels of abstractions, allowing separation of concerns and better complexity management.
3. It allows analysis of certain system properties.
4. It creates parts (technological implementations) automatically via transformations.

3 Motivating Example

As mentioned before, service adaptation helps to solve various problems due to, e.g., interaction protocol or QoS mismatches, or service failures. These issues can be found in all application domains that require the inclusion of new application features, the scalability of services, or their replacement with alternative ones.

For example, in order to target new clients, the creators of an e-commerce application platform decide to extend it through the inclusion of new features (e.g., inclusion of a one-click checkout protocol). The interaction protocol required by the new clients might be different from the old one. In this case, the application can be adapted to support both interaction protocols so as to sure that the enhanced version of the application (transparently) offers the required functionality to all its clients—both old and new. Another example is an online video-sharing application that has to allocate resources in order to scale up with an increasing number of online clients. Such an application has to provide a flexible adaptation to the resource availability (e.g., upload and download bandwidths) and protocols (e.g., routing protocol for data streaming) that allows it to support a desired quality of the user experience. The adaptation may depend on the number of active users, their subscription plan, their current activities, and their physical location.

In this paper, we focus on an example that illustrates the adaptation of an authentication service due to the failure of one of its components. This example is based on the experiences of Tech Media Telecom Factory SL (TMT),[1] a

1. http://www.tmtfactory.com/.

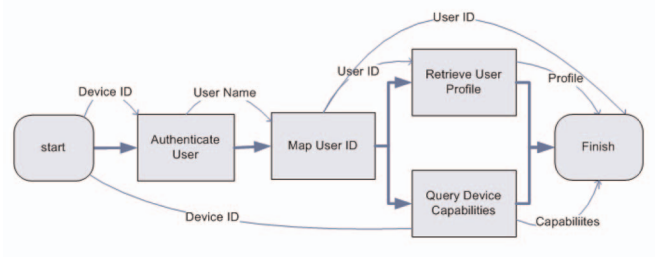

Fig. 2. TMT scenario for admission and authentication.

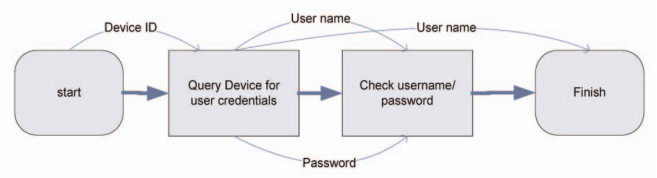

Fig. 3. Initial authentication action.

partner in the ALIVE project. The scenario defines a tourist recommendation system that interacts with users via a distributed, interconnected system of smart terminals. These terminals provide information, recommendations, and bookings for tourist services such as restaurants, cinemas, or events in a geographical area based on users' personal preferences. The system consists of a centralized control service (core system) and a large number of client devices. Each client device (smartpoint) is remote and is connected to the core via the Internet. The smartpoints act as service consumers (e.g., of travel information services) and service providers. They expose functionality as a set of services that allows the core system to interact with the user and vice versa.

In the scenario, a series of user interactions occurs and is mediated by the system via an orchestrated dialog in the form of a workflow (see Fig. 2). In this example, we focus on a particular aspect of the overall system functionality, which deals with user admission and authentication.

The scenario is as follows: In order to gain access to the recommendation system functionality, the workflow illustrated in Fig. 2 is invoked within the system (i.e., initiated by some user interaction at one of the smartpoints). The workflow takes a device identifier (i.e., a URL which refers to the specific smartpoint) as input and first queries the device for the user's authentication details that, if successful, yields a user name. Then it is mapped into an internal system identifier for that user via an internal system service/component. Subsequently, the system queries a profile service to retrieve (or create) the user's profile and queries the user's device for its capabilities. The user identifier, user profile, and device capabilities are then returned by the workflow.

As deployed, the authentication action is a simple challenge-response call, which queries the user's device for a user name and credentials (see Fig. 3). These are then checked internally against an internal database, where they are mapped into an internal system identifier, corresponding to the user's identity in future transactions.

We express the specification of the authenticate call (see the Authenticate User task in Fig. 2) using a simple (OWL-S [11]) process with the following inputs, outputs, preconditions, and effects:

$$\begin{aligned}&\text{Preconditions: none,}\\&\text{Inputs: DeviceID uri,}\\&\text{Outputs: UserName user,}\\&\text{Effects: } \mathrm{is_authenticated(user)},\end{aligned}$$

where DeviceID and UserName denote OWL classes [19] referring to the URI of a particular device and the name of a user, respectively; user and uri denote parameter identifiers for input and output parameters of the process; and *is_authenticated(user)* is a semantic predicate (expressed in SWRL [20]), which indicates that following the successful execution of the process, the user is successfully authenticated at the device in question.

In this motivating example, the adaptation required is the replacement of the authentication call with an alternative system. This may be necessary because of 1) the failure or removal of the existing authentication database or 2) because of changes in the requirements of the call.

Where the authentication service ceases to be available, the whole authentication workflow will fail persistently. We assume that such a failure is detectable. In this case, the system would try to find an alternative to the failed service—that is, a service which takes a username and password and asserts that the user belonging to the user name is authenticated. When this process fails, the entire subworkflow becomes invalid and a new workflow fulfilling the same requirements must be constructed.

Where the requirements have changed, we assume that an alternative means of authentication is available and operates using a different underlying user-system protocol to the original authentication mechanism.

The alternative protocol is summarized as follows:[2] The protocol is initiated by a calling service that interacts with the user who has to be authenticated. The calling service queries the user for a personal URL (e.g., http://ist-alive.eu/thanos), which acts as an identifier from the perspective of the user. The system then performs a query using this URL (typically by fetching the URL itself and retrieving some header information) in order to determine the authentication service that should be used to authenticate the user. The user's Web browser (or a browser-based interface on the target device) is redirected to the (external) authentication service, which then prompts the user for authentication. This redirection includes a return URL, which is owned by the calling service. Following successful authentication, the authentication service redirects the user back to the specified return URL (within the calling service) with a cryptographically signed token indicating the user's identity. Finally, this token is checked by the calling service before allowing the authenticating user to proceed.

Fig. 4 shows the workflow for a client, where the lighter (blue) boxes indicate protocol-specific calls and darker (green) boxes indicate application-specific calls, which must be filled by the client application invoking the protocol. The

2. The protocol here is based on the OpenID [21] authentication system (http://www.openid.org); however, the overall flow of interaction is similar to that found in other Web redirection protocols such as Shibboleth (http://shibboleth.internet2.edu) or Microsoft Passport (http://www.passport.net).

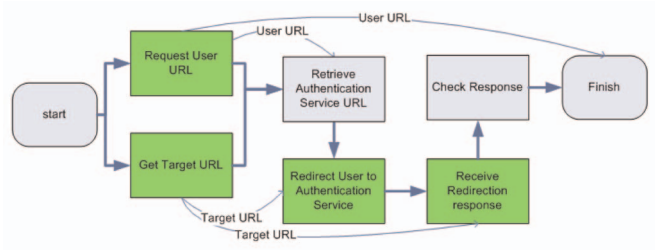

Fig. 4. OpenID, alternative authentication mechanism.

outcome of the protocol is a User Resource Identifier (URL) and an assertion that the user with that URL is authenticated (as with the previous protocol). The process of adaptation discussed in the remainder of this paper relates to the process by which the initial authentication process is replaced by the redirection protocol described below.

4 Proposed Solution

This section introduces an approach to the development and adaptation of service-oriented applications based on semantic descriptions and model-driven techniques. The proposed development process is based on the concepts of 1) model-driven engineering, 2) semantic technologies for selection and reasoning, and 3) service adaptations based either on service templates or direct modifications.

Our approach considers service adaptations primarily as substitutions, conversions, composition, or direct modifications of the service properties and parts. In addition, they are distinguished into the following:

- **Planned**: when the semantic reasoner matches and selects appropriate concepts and services that parameterize prebuild service templates. As a result of this parameterization, an adapted service is created and deployed.
- **Spontaneous**: when the adaptation is performed without the use of predefined service templates but with the direct modification of properties and parts of the service model.

In the first case, the adapted service is created as a result of model synthesis (service model + template + matched services), and in the second case, as direct modification of the model's properties and parts. Both service adaptations are performed with model-driven means and accompanied with semantic reasoning tools.

The development process involves the following steps:

1. modeling and metamodeling,
2. automatic creation of implementation artifacts,
3. execution and monitoring,
4. semantic analysis, and
5. application of adaptations.

Fig. 5 outlines graphically the process steps.

Step 1: Modeling and metamodeling. Model-Driven Engineering is used for 1) the specification of service and template metamodels, 2) the creation of special purpose editors capturing services, and service templates with models abstractions, and 3) the specification of adaptation

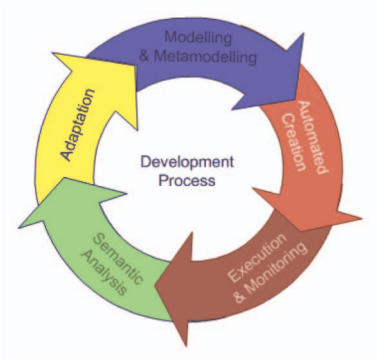

Fig. 5. Proposed development process.

mechanisms that will perform the model synthesis and transformation process.

Step 2: Automated service creation. Service implementations are automatically created after applying predefined transformations among source models (e.g., service models) to target languages (e.g., Java Web services). MDE is also used to bind specific services and parameters to predefined service templates to create and deploy new adapted services within a service-oriented environment.

Step 3: Execution and monitoring. The actual execution and monitoring of services is implemented using execution engines for Web services and special-purpose-built components that observe the service interactions, state transitions, failures, and environmental properties, against certain conditions and requirements. Details of the actual execution and monitoring mechanism are left outside the scope of this paper. However, for more details, we refer the reader to [22].

Step 4: Semantic analysis and selection. Semantic analysis and selection of services are based on ontological descriptions and reasoning tools. Service selection refers to the process of locating an existing service based on the description of their functional or nonfunctional semantics. Within MDE, the service models are annotated with semantic descriptions that are transformed to OWL-S descriptions. By using standard means of logic reasoning, it is possible to determine the semantic relevance (matching) of a service to particular requested service characteristics.

Step 5: Service adaptation. Service adaptations are performed on service models, representing real service entities, which are created via automated transformations from service templates. The adaptation itself is a process of substituting one service for another, converting and composing new services from existing ones, as well as applying direct modifications on service properties and parts. In particular, composition and conversion are supported by the definition of service templates at design time. Third-party adaptation techniques can also be employed to support the semiautomated generation of service templates.

In general, the introduced benefits are in the formulation of service templates that provide patterns of service adaptation with metamodels, the capture of templates at design time with model abstractions by using specific purpose-built editors that are created semiautomatically, and the automatic creations, at runtime, of the adapted

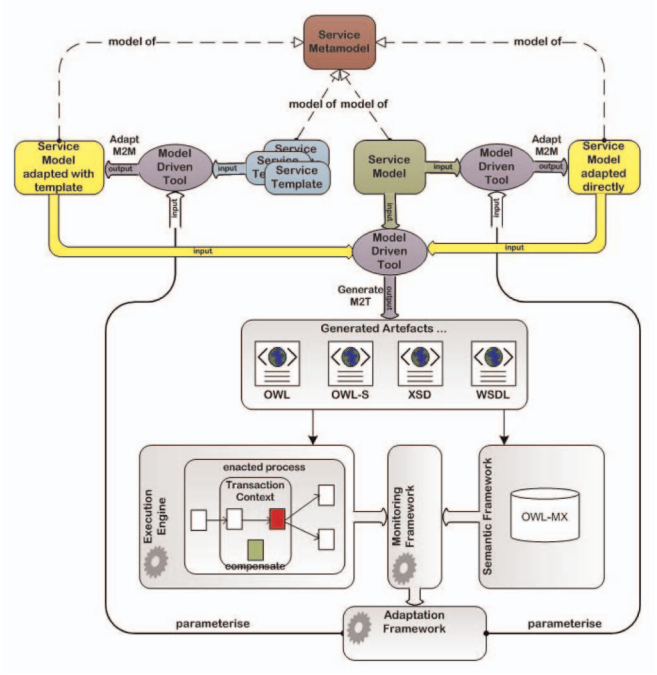

Fig. 6. Components realizing the development process of our approach.

services by parameterizing the service templates with services that are dynamically selected.

The proposed approach and methodology is illustrated using the ALIVE framework (see Section 5). In particular, Fig. 6 depicts the primary components that will realize the proposed development process within a service-oriented framework, in this case, in ALIVE.

In brief, the development process is initiated by the "modeling and metamodeling" step, where a service metamodel is formally specified using the Ecore [23] metamodel language (Section 5.1 explains in detail the metamodel segments used in ALIVE). Similarly, metamodels are used to specify service templates (see Section 5.3). Service templates are designed to accommodate specific types of problems. They combine and convert the existing services based on the input and output parameters and the pre and postconditions of services. When these templates are bound to specific services either at design or runtime, they form adaptors. Adaptors expose the new services either as a result of composition (composite services) or conversion (wrappers). Next, based on the metamodel, a graphical editor is created so that specific service models can be captured (instantiated) by a software designer (see Section 5.2).

At the "automated creation" step, predefined model-to-text (M2T) transformations are applied on these models by model-driven tools to automatically generate associated syntactic and semantic implementation artifacts (in the form of, e.g., WSDL or OWL-S) descriptions, service implementations (in the form of Java skeletons), deployment descriptors (e.g., WSDD for Axis), and published registry entries (e.g., UDDI). In this way, the metamodel fully supports the development of semantic service-oriented applications with model-driven means.

At the "execution and monitoring" step, the deployed services are executed by the execution engine and monitored by the monitoring framework that is composed of components observing the service enactment. Once a problem or an error occurs, the enactment engine will try to first handle the error with the in-place mechanisms (e.g., error handlers, roll-back activities, transaction context, etc.) and second generate a relevant event that will be passed to the monitoring mechanism that will try to resolve the problem via an adaptation process.

At the "semantic selection and analysis" step, the semantic framework (composed of matchmakers and reasoning tools) will analyze the events generated and select via matchmaking the templates or other suitable adaptations, offering an alternative service or a modified service to address a stated problem (see Section 5.4).

Finally, the "service adaptation" step refers to the process initiated by the adaptation framework (a tool that utilizes a model-driven transformation approach) to synthesize the new adapted services from service templates and services identified at the matchmaking process with an M2M transformation (see Section 5.5).

5 Solution Applied to ALIVE

In this section, we demonstrate how our approach is applied within the framework of the ALIVE project [10] to perform an adaptation of the authentication service (see motivation example). The approach is realized using tools such as Eclipse editors, semantic tools (OWL-MX) [24], and transformation (QVTO) [25] tools.

The ALIVE project proposes a new approach to the engineering of service-oriented systems based on coordination and organization mechanisms often seen in human and other societies. To achieve this, it performs the following tasks:

1. develops an advanced framework for application development, deployment, and management in service environments,
2. utilizes model-driven engineering techniques and tools,
3. provides a dynamic methodology for service design, adaptation, and maintenance, and
4. supports an alignment with other emerging architectures and standards [10].

5.1 Service-Oriented Metamodel Engineering

In ALIVE, the service-oriented paradigm is conceptualized and formalized with a metamodel specification (Ecore). The service-oriented metamodel (see Fig. 14 in the Appendix) defines and supports fundamental service concepts and characteristics such as service providers, consumers, and registries, allowing the self-description, publishing, and discovery of services via corresponding metamodels. At runtime, the described metamodel capabilities become implemented functionalities of actual executable components that are derived via model-driven transformations. Due to the space limit, we describe only the elements and supporting infrastructure that are related to the adaptation of services.

The ALIVE services metamodel is structured into syntactic, semantic, architectural, and service template parts. Fig. 14 in the Appendix depicts parts of the service-oriented metamodel. In the diagram, the service-oriented concepts/entities are depicted as EClasses (rectangular)

whereas the relationships among concepts are illustrated with links (EReferences).

The **syntactic part** of the metamodel defines the elements required to specify a service's functionality as an exposed interface. A service is likely to have operations with input and output parameters, types of generated faults, as well as protocol details and interaction styles. From the syntactic part of the service model, a service's description (e.g., WSDL) could be automatically generated and exposed to a registry (e.g., UDDI), so it can be located and used by other services.

More specifically, a *Service* represents a modular functionality unit, has a textual name, description, and namespace in which it is defined and available. A specific service consists of a concrete *Endpoint* from where it can be invoked and an abstracted *InterfaceDescription* providing the functional signature of the service. *Endpoints* capture the location of the service together with information *Binding* that provides protocol details (such as encoding and style to be used) and transport mechanisms (such as SOAP over HTTP). The *InterfaceDescription* has a name and a number of operations (*OperationDescription*), referring to the functionalities exposed by the service. Each *OperationDescription* has ordered inputs, outputs, and faults, all of which are of type *Message*. A *Message* refers to the data types and concepts exchanged during the service invocation. They refer to XML Schema *Elements* and *ComplexTypes* that can be either defined inline (within) the service description or separately.

The **semantic part** of the metamodel specifies what the service does, by providing ontological annotations for various service elements. It has *ServiceProfile, ProcessModel*, and *ServiceGrounding* elements. The semantic parts of the model are mapped to the corresponding parts of the OWL-S specification [11], so the semantic description of the service can be generated. The semantic part is used by the matchmaking process to select services based on certain criteria.

A *ServiceProfile* provides a higher taxonomic description of the service, so it can be selected by category or other nonfunctional properties. The semantic functionality of a service is described by the *ProcessModel*. The *ProcessModel* provides a semantically grounded description of a service's invocation in terms of the *inputs*, *outputs, preconditions*, and *effects* (IOPEs). It gives a high-level interpretation of a given service call, where each *input* or *output* corresponds to an ontological concept such as OWL [24] and where each *precondition* and *effect* relates to a rule-based language such as SWRL [20]. For a given service, *ServiceGrounding* binds the syntactical parts of its interface description to the corresponding semantic (ontological concepts) parts of its *ProcessModel*. So, it enables a service to be invoked accordingly to its semantic descriptions and conditions.

The **architectural part** of the metamodel captures the elements used by the execution framework such as the enactment and monitoring components. In turn, the enactment component may be related to exception handlers and transactional coordinators that perform transactional protocols within a specific context. In this case, the transactional model employs WS-TX concepts to generate the transactional context for the processing for the enacted services. Actual details of the execution and monitoring mechanism are outside the scope of this paper, so they are not shown in detail.

The **template part** of the metamodel provides the elements required to specify service templates as means of service adaptation. A service template acts as a collective description of abstracted process models that resolve to a specific functionality or goal. Similar to services, templates are stored in template repositories. They are discovered by matchmaking components that implement them with concrete services.

Each *ServiceTemplate* has a *URI* uniquely identifying the template. It also has parameters of type *Concept* (referring to ontological resources), a *TemplateFlow* and an exposed *AbstractProcessModel*. The exposed *AbstractProcessModel* defines the abstracted process model type for an adapted service. An *AbstractProcessModel* becomes concrete (bound) once the template is instantiated via an adaptor (*ServiceAdaptor*), so it can be exposed as a service. It consists of abstracted parameters (inputs, outputs) and conditions (effects and preconditions). The *TemplateFlow* provides a container in which abstracted partner processes (*AbstractProcessModels)* are specified. *AbstractProcessModels* of a *TemplateFlow* are connected with *Links* via the *source* and *target* associations. A *Link* connects two or more *AbstractProcessModels* to specify ordered interactions among participant partner processes. *Conditions* are specified on *links,* which, once they are satisfied, activate the *target* of the *link*. A number of *MapConcept* elements can be specified on *links* in order to map output variables of one process model to input variables of another, for example ($\texttt{outConcept}_1 >> \texttt{inConcept}_2$, $\texttt{outConcept}_2 >> \texttt{inConcept}_1, \ldots$). In this way, multiple *Concepts* can be passed from one process to another. With *Links,* it is possible to specify sequences, flows, loops, and conditional execution, similar to the links in WS-BPEL [26].

A *ServiceAdaptor* provides a particular implementation of a *ServiceTemplate* using bindings from the abstracted process models (*BindProcessModel*) and parameters (*BindTemplateParmeter*) to actual process models and parameters of available services. An actual adapted service is *exposed* for use, fulfilling (*implementing*) the requirements of the service template. A *BindTemplateParmeter* is an element of the adaptor that maps abstracted template parameters, which are related to ontological concepts, to specific service parameters and concepts. Similarly, the *BindProcessModel* is an element of the adaptor that maps the abstracted process models of a template to actual process models of real services.

5.2 Modeling the TMT Case Study

This section presents how the TMT motivating scenario is captured as a model (see Fig. 7) of the ALIVE metamodel, with an Eclipse graphical editor created with the Graphical Modeling Framework (GMF) [27].

The user may invoke the *findEvents* operation of a TMT Interface of a TMT Service. The service internally triggers a workflow that authenticates the user and identifies his/her preferences. The workflow, which is part of the ALIVE's coordination level, involves four services: *Direct Authentication*, *Identity Map Service*, *User Profile Service*, and a *Device Capability Service*. Multiple workflows or services may be specified—in this example, *OpenID Provider* is available in the system, though not part of the *findEvents* workflow. The services are registered with a *Service Directory*, so they can be located and dynamically invoked by other services.

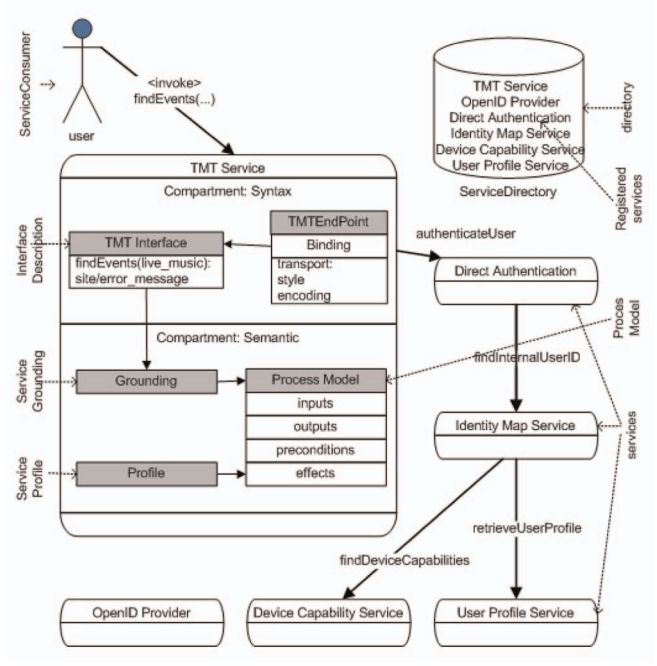

Fig. 7. TMT scenario service model.

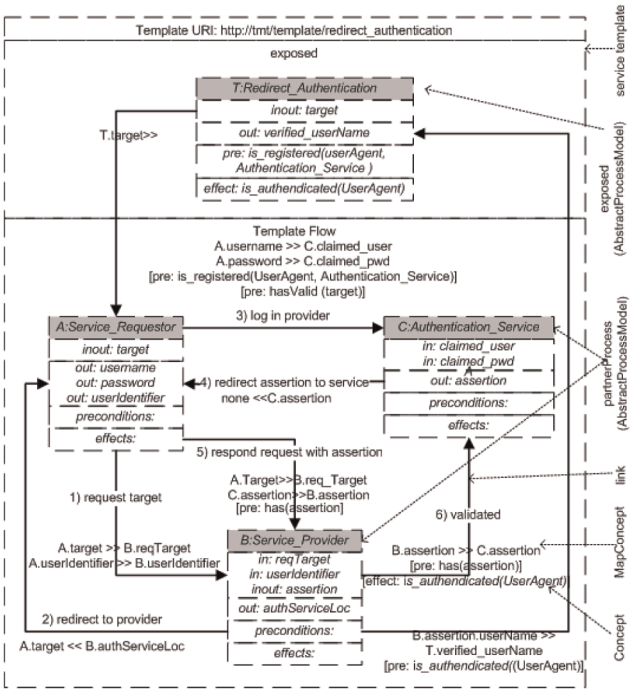

Fig. 8. Specifying a service template for redirecting authentication.

A *Service* model consists of two compartments. The upper one provides the syntactic description of a *service* with *interfaces*, *endpoints*, and *bindings*, whereas the lower part contains its semantic description with a *Grounding*, *Profile*, and *ProcessModel*. The compartments are not expanded for the other services, so their internal parts are hidden. Similarly, a *ProcessModel* has compartments for *inputs*, *outputs*, *preconditions*, and *effects*.

5.3 Specifying Service Templates

In this section, we illustrate how to specify, via a service template model, an alternative authentication mechanism that provides a general authentication solution, allowing the user (agent) to be authenticated via redirection to an external authentication authority (refer to motivating example). Should the direct user authentication fail, the adaptation module will attempt to find available services satisfying the service template's requirements (inputs, outputs, preconditions, and effects) and expose a new service (e.g., OpenID Service Provider) replacing the previous, so the authentication can continue. Fig. 8 depicts the service template model for the redirection authentication.

The service template editor depicts service templates as a rectangle with two compartments. The lower compartment contains the template's flow, which connects the *AbstractedProcessModels* of participant services. Each of the *AbstractedProcessModel* services such as *Service_Requestor* contains compartments specifying the abstracted *inputs*, *outputs*, *preconditions*, and *effects* with concepts and rules. Next, on the links, the output concepts are matched with input concepts in the form of `A.target >> B.reqTarget`. Preconditions and effects are applied to make the flow deterministic. The upper compartment contains the *AbstractedProcessModel*, exposed by the template and which is the result of interaction among the participating services.

In this case, the abstracted partner processes involved are those of *Service_Requestor*, a *Service_Provider* and an *Authentication_Service*. The flow is initiated when the *Service_Requestor* makes a request for a *target* resource to a *Service_Provider* by using his/her *userIdentifier*. The *Service_Provider* responds with the *location* of the *Authentication_Service* to be used by the *Service_Requestor* for authentication. By using this location (*target*), the *Service_Requestor* makes a contact with the *Authentication_Service* by providing a *username* and *password*. The login phase is performed as before, however, now it is validated by an external authority. In response, the *Authentication_Service* creates a signed *assertion*, which is sent back to the *Service_Requestor* to be forwarded to *Service_Provider* as a new signed request. The *Service_Provider* validates the signed request with the *Authentication_Service* and if it is valid, the validated *username* in returned as output of the flow.

5.4 Template Matchmaking and Selection

Existing matchmaking approaches compare queries against single-service descriptions. Templates add another dimension to this approach in that we must decide whether or not a template should be used for a given query, which template to use and which partner services best fit to provide a final composed template match.

In order to accommodate the matching of templates against queries, we use an extended hybrid matchmaking approach described as follows: Each matchmaking phase starts with a service query consisting of a set of IOPEs and a service profile. We assume the presence of an existing hybrid matching function *MatchSingle*, which takes a query and determines the most appropriate single service for that query using a number of matching metrics, or returns nothing if no single service is found. Candidate template

```
MatchMaker(Q,D = 0):BEGIN
   If D > MaxDepth: BEGIN
        Return [28]
   END
   ST_Q := BuildSingleton(Q)
   CT := {ST_Q} + SearchTemplates(Q)
   BTS := {}
   For each template T in CT: BEGIN
        BL := {}
        SS := {}
        For each service slot L  in T: BEGIN
           SS += MatchSingle(QL)
           SS += MatchMaker(QL, D+1)
           If SS is empty: Skip to next template
           SL := Rank(SS)[0]
           BL += {L -> SL}
        END
        BTS += {BindTemplate(T,BL)}
    END
    Return first (Rank(BTS))
END
```

Fig. 9. Template matching algorithm.

```
engine = TransformationDef("RedirectAuthService");
engine.executeTransformation
(TMTScenario::ServiceModel,
 RedirectTemplate::TemplateModel,
 mapServices::Map(abst::Service, conc::Service)
 mapConcepts:: Map(abst::Concept, conc::Concept)
, ...);
```

Fig. 10. Adaptation module triggering the transformation description to createAdaptor with parameters resolved by matchmaking.

sets are selected using the *SearchTemplates,* which select possible candidate templates by performing a keyword search based on the ontologies and terms referred to in a query and those of the known templates stored in a common repository. This operates in a similar fashion to the candidate service selection process used in conventional matchmaking described in Section 2.4. In order to accommodate the case where a query may be matched by a single service without the need for template adaptation, we define a function *BuildSingleton* that constructs a special *singleton* template based on a query consisting of a single slot that corresponds to the query itself such that matching against this template is equivalent to searching for a single service which satisfies the query.

The template matchmaking process (shown in Fig. 9) operates as follows: If the matchmaker has reached the maximal defined depth, it returns with no results. We then build a singleton template based on the query and combine this with the set of templates returned by *SearchTemplates* to give a candidate template set *CT*. For each candidate template in this set, we then iterate over each of the unbound slots (*L*) and first try and find a single service that matches the slot (*MatchSingle*) before repeating the template matchmaking process for the given slot. Where a service result is found for the given slot (*SL*), we store the possible binding in the map *BL*. In the case that a slot cannot be satisfied, we skip to the next template. When all slots are satisfied, the template and its bindings are stored as a candidate match in the set *BTS*. Finally, all candidate matches are ranked and the most appropriate match is returned.

```
//map definition to create a ServiceAdaptor model
mapping createAdaptor(
  in template: ServiceTemplate,
  in pm: Tuple(absProcessModel:AbstractProcess
             Model),conProcessModel:ProcessModel),
  in cm: Tuple(absConcept:(Concept),
              conConcept:(Concept))
):ServiceAdaptor {

//create Service Adaptor model
result := object ServiceAdaptor{
 name := template.URI + '_Adaptor' ;

//bind the (abstract <-> concrete) processmodels
of the partner processes wirthin the TemplateFlow
 bindProcessModel += object BindProcessModel{
  concrete := pm.conProcessModel->at(index);
  abstract := pm.absProcessModel->at(index);
 };

//assign the implemantion template
 implement := template;

//create actual service model that will be ex-
posed. Later will be transformed to WSDL
 expose := object Service{
  name:= "ServiceExposed";
  describedBy := object ProcessModel{…};};

//bind the (abstract <-> concrete) processmodels
for the exposed adapted service
 bindProcessModel += object BindProcessModel{
  concrete := result.expose;
  abstract := template.exposed;
};
```

Fig. 11. Adaptor and exposed service is created via transformation.

As each bound template has an exposed process model, we can apply the same matching metrics to bound templates as we do to single services. It is also possible to consider metrics, which take into account the structure and properties of the services of subtemplates that make up the template itself. Splunter et al. [29] suggest a number of such metrics such as preferring singleton matches (single services) over templates, preferring template matches with fewer slots (dependent services), or preferring templates with the highest aggregate match quality for the underlying services.

5.5 Applying Model-Driven Adaptations

In this section, we present how the redirected authentication template is parameterized on the fly from semantic tools in order to create an adapted (collaboration) service via model transformation.

During semantic matchmaking, the semantic tools will select the appropriate services and concepts satisfying the implementation requirements of a service template. Following, the adaptation module triggers the corresponding transformation at runtime (see Fig. 10) and passes the selected elements to the transformation description (see Fig. 11).

The transformation definition (`RedirectAuthService`) performs the adaptation on the `TMTScenario` and the `RedirectTemplate` models. The parameters passed to the

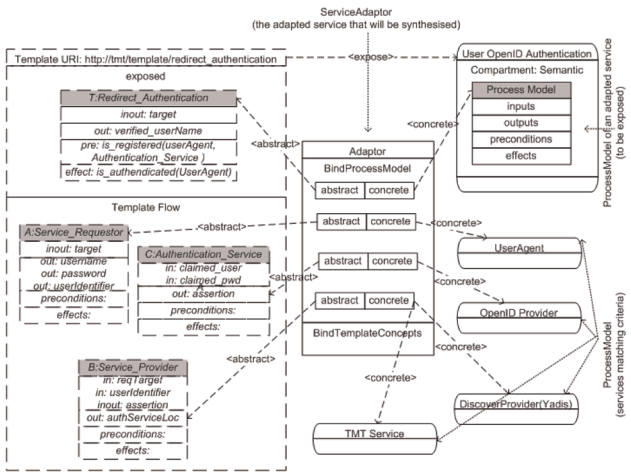

Fig. 12. Adapted service (exposed) is created via a transformation.

transformation engine are 1) a map with the abstract and concrete `Services` and 2) a map with the abstract and concrete `Concepts` matched during the matchmaking process. The actual transformation is performed with tuples that are standard OCL elements [30], expressing maps where `Services` and `Concepts` are actual model elements.

Within the transformation definition, a `createAdaptor` mapping (see Fig. 11) is applied by taking as inputs the `template` for the adaptation, a `pm Tuple` (map) between the `AbstractProcessModels` and `ProcessModels`, as well as `cn Tuple` between the abstracted and concrete `Concepts`, thus producing the `ServiceAdaptor` model element.

Once the transformation is completed, an adaptor model and an exposed service are created. Fig. 12 depicts the creation of the adaptor and service.

Next, a model-to-text transformation (see Fig. 13) creates the actual code (e.g., WSDL, WS-BPEL) and deploys the service for usage to its environment. For example, the following extract shows how the *Service* and *Port* elements of WSDL are created for the adapted service. The transformation is applied with the Acceleo[3] script transformation language. In particular, the code snippet defines two scripts (mappings), the `mapService` and `mapPort`. The former creates a WSDL service with name and description and which triggers the latter in order to create within the service a WSDL port for each endpoint defined in the model, with name, binding, transport protocol (SOAP or HTTP), and location.

Similarly, model-driven adaptations can be applied directly to modify the service's properties and parts, without the use of service templates. For example, input or output types of interfaces may be changed and additional preconditions and effects may be imposed dynamically. When compared to template adaptation, this kind of adaptation is less complicated. However, as previously, the transformation process is directed from the adaptation and matchmaking process to create, delete, or modify existing modeling elements of the service model and provide their implementation parts.

3. http://www.acceleo.org.

```
//For every exposed service create a Service WSDL
<%script type="Service" name="mapService" %>

//assign exposed model's name and description to
service
<service name= "<%name%>" >
   <documentation> <%description%></documentation>
//create WSDL port
   <%mapPort(prefix)%>
</service>

//create WSDL port
<%script type="Service" name="mapPort" %>
//for every endpoint
<%for (endpoint){%>
//assign the port name and binding
<port name="<%name%>" binding= <%binding.name%>">
<%if (binding.transport.equals("SOAP")){%>
    <soap:address location="<%location%>"/>
<%}else{%>
  <%if (binding.transport.equals("HTTP")){%>
    <http:address location="<%location%>"/>
  <%}%>
<%}%>
</port>
```

Fig. 13. An acceleo model-to-text transformation script generating the WSDL service and port elements for the exposed service.

6 Critical Discussion

The core of our service adaptation approach is based on the modification of service models, using model-driven engineering, service templates, and semantic matchmaking based on ontological descriptions. The synergy of these technologies introduces many benefits to our adaptation approach, such as:

1. *Abstraction*: the level of abstraction is raised via the use of models and the specification of adaptations, captured as template models (patterns, types) of common behaviors.
2. *Automation*: service implementations are automated by the application of transformations and automated semantic matchmaking.
3. *Usability* is enhanced with the use of Eclipse editors and graphical notations, the application of predefined transformations, and provision of libraries of service templates.
4. *Effectiveness* of development as transformations are pretested.
5. *Interoperability*: connectivity between external tools and formats is enabled through the use of open standards.
6. *Extensibility*: third-party adaptation techniques may be employed to support the semiautomated generation of service templates, which may be needed to cope with complex adaptation scenarios that require solving several issues (e.g., behavioral and QoS mismatches).

In general, dynamic software adaptations are supported by reflection and late binding. *Reflection* is the process where a system can observe and modify its own structure and behavior [31]. The observation property is referred to as introspection, and the alteration property as interspection. MDE supports both reflection properties natively, via its architectural design and by providing a powerful API for

metaobject management. *Late binding* refers to mechanisms where decisions can be resolved at a later point in time such as runtime. Late-binding capabilities are also possible for MDE as its actual implementation is based on programming languages such as Java. So, MDE supports both properties to enable dynamic software adaptations in a similar way as with object-oriented and component techniques. When compared to other approaches and methodologies (see Section 7), we provide an adaptation process, which is based on model abstractions and where service templates represent abstracted solutions for specific types of problems.

The template matchmaking component presents a challenge with respect to the complexity of finding suitable configurations to adaptation problems in a reasonable time. The search space for any automatic configuration problem in a service-oriented system will inevitably be large, as a large number of services may be present as candidate matches for a given query. Template matching enlarges this space further by allowing for multiple candidate templates and multiple candidate services for slots in those templates. To a certain extent, this complexity can be curtailed by limiting the scope of the matchmaking search by terminating the search after a given period of time, or by restricting the depth of template query matches. The crisp, semantic component of the matchmaker (determining whether the IOPEs of a given query satisfy a given query) lends itself well to current state-of-the-art planning approaches in the artificial intelligence community such those based on SAT solvers and nonmonotonic logic programming techniques. Within the project, we are developing a constraint-based search system, which uses the Answer Set Programming [32] nonmonotonic logic programming language to encode template configurations and service descriptions in order to facilitate more efficient template-based matchmaking.

Generally, the ALIVE framework and methodology are evaluated step-by-step by applying case-study-based evaluation against three large-scale scenarios from different domains, provided by our industrial partners. The motivating example presented in this paper is part of one of those scenarios.

Finally, we advocate that our suggested methodology is pragmatic as it is based on existing and well-established development methodologies, Web and semantic technologies, and is applied in realistic examples.

7 Related Research Work

In this section, we consider active research from the fields of service composition and adaptation supported by model-driven techniques or other means.

Model-driven approaches and frameworks supporting adaptation include DIVA, MADAM, and Rainbow [33], [34], [35]. In DIVA [33], an application is modeled at design time with a base model (containing the common/core functionalities), a set of variant models (capturing the variability of the adaptive application), and an adaptation model (specifying which variants should be used according the rules and current context of the executing system). At runtime, the models are processed by model composers that produce the system's configuration. By comparison, ALIVE takes a more lightweight approach as it is not based on a dedicated metamodel for an adaptation framework, but rather uses parameterized adaptation templates to adapt a specific type of functionality. Selection and substitution is based on semantic matchmaking of services in the application domain.

MADAM [35] achieves runtime adaptation through the use of architectural models. Comparatively, their interest is in adapting and configuring architectures of mobile adaptive systems, while adaptations in ALIVE are performed on services, not their underlying middleware, so modifications are applied on service's behaviors, structures, and organizations.

Rainbow [34] provides an adaptation framework based on an abstract architectural model to monitor runtime properties to accommodate, for example, resource variability and system faults. Similarly, in ALIVE, our architectural model represents the execution environment that monitors and evaluates certain rules attached to service models, based on which a model-driven adaptation is triggered. However, in both cases, adaptations are specified and implemented differently.

In addition, there are several approaches and techniques dealing with the broader concept of service adaptation. For example, Chang et al. [7] propose service adaptation based on four types of service variability, which can be implemented in a typical Web service environment. These include the following:

1. workflow variability (different invocation orders),
2. composition variability (more than one service can be bound),
3. interface variability (interface signatures, when their semantics does not match), and
4. logic variability (concept variation, semantics).

It also identifies seven adaptation methods, namely:

1. delegation,
2. selection,
3. plug-in,
4. external profile,
5. mediator,
6. transformer, and
7. enhancer.

These mechanisms (patterns) are derived from Object-Oriented Programming (OOP) techniques. An adaptation manager resolves these four types of service variability. In our case, the adaptation manager utilizes the semantic matchmaker and triggers model-driven transformations to perform adaptations.

Another framework for the dynamic customization of services is proposed by Sam et al. [36]. In this case, customization is based on syntactic, semantic, and constraint comparison of input and output types between requested and available services using the LARKS [28] Web description language (a predecessor to current semantic Web service languages such as OWL-S). In their approach, only sequential compositions of services are considered, whereas we permit arbitrary structural compositions as specified by template descriptions.

Jiang et al. [37] address the notion of reuse in Web service development. Reusability is supported via categorization of possible variation points to support a family of services having common architecture and functionalities. Management of variation points is based on a pattern-based approach. In contrast, our approach is not based on variation points but service templates.

There are also techniques considering service adaptation as substitutions to facilitate high availability of services. For example, Birman et al. [38] propose a set of extensions to the Web services architecture that allows application developers to enhance the reliability and availability of service-based applications. While their approach somewhat tackles adaptation issues at the lower layers of the Web services stack, we aim to define an adaptation framework that tackles adaptation issues at business process layers such as behavioral or QoS issues. These two approaches complement each other and may be used together to provide highly adaptive service-based applications.

In addition, Liang et al. [39] propose a novel Web service matching technique to support service substitutions when using different service domain ontologies. The matching employs a term categorization step and a rule-based service matchmaker. The former selects terms in the service descriptions, while the latter applies semantic rules to check whether the compared services are equivalent with respect to the categorization results. The approach then selects the best match as candidate for substitution. In our approach, the runtime substitution of services employs a matching process that assumes a common ontology for service descriptions. While our approach will match fewer services, we believe that using a common ontology minimizes the chances of false positive matches, and hence, it allows us to have a better service matching precision.

Emerging approaches to adaptation of services based on compositions include Sheng et al [40], who present a system supporting configurable and adaptive composition of Web services. It is based on three core services; a coordination service, a context service, and an event service that automatically schedule and implement user-configured adaptations at runtime. Composition is achieved with a process schema similar to a UML statechart. In our case, the equivalent concept is that of abstracted template flow, which is dynamically parameterized by concrete services and concepts.

Other emerging techniques for flexible software adaptation are based on Aspect-Oriented Programming (AOP) [41]. For example, Hirschfeld and Kawamura [42] address dynamic service adaptation by using the aspect modularity construct to represent units of change. Language reflection and dynamic aspect-oriented programming allow the adaptation of services when it is required. When compared to our approach, we use model adaptation where modularity is represented by the service model element itself.

Another aspect-oriented approach for service adaptation is by Kongdenfha et al. [43]. Here, the approach is based on a taxonomy of mismatch types on the invocation signature based on input types, their ordering, as well as flow of exchanged messages. In our case, we do not provide an explicit classification ontology; however, this is possible by either extending the ontological description of our service domain or by reusing existing ones.

Finally, Hibner and Zielinski [44] propose a semantic-based dynamic service composition and adaptation framework. Their work is based on the Web Service Modeling Ontology (WSMO) [45] to semantically compose services using backward chaining reasoning. WSMO provides the semantic descriptions and deals with interoperability between different elements via mediators. There are four types as follows:

1. ontology mediators,
2. Web service,
3. goal, and
4. services and goals.

An Enterprise Service Bus (ESB) executor creates the complex service. The paper also distinguishes two kinds of adaptation 1) external (as service composition) and 2) internal (within a service). In our approach, workflow composition is based on GPGP planning [46]. In the role of ESB executor, the adaptation module performs the transformation and creates/adapts the service. The concept of external adaptation corresponds to that of service templates, and the internal adaptation corresponds to the direct adaptation of the service model.

8 Conclusions

In this paper, we have presented a model-driven approach for the dynamic adaptation of service- and business-oriented applications to cope with implicit and explicit changes to their requirements and the environment. Adaptation is an essential property for such long-lived enterprise systems, which need to achieve higher levels of autonomy and handle unexpected problems to be continuously running. In the context of this paper, service adaptations are performed on abstracted service models as a result of a transformation process. This has the following main advantages: First, it raises the level of abstraction through which the designer can reason about adaptations. Second, predefined transformations and automation reduce the extent to which errors can be injected into the development process. Third, the usability is enhanced through the provision of purpose-built editors that facilitate the methodology steps. Finally, the approach can be used in conjunction with third-party adaptation techniques to semiautomatically generate service templates from which one can deploy service adapters.

After the models are adapted through transformation, adaptations are then reflected back to the corresponding service implementations. Service adaptations are supported in two ways: 1) using service template specifications that capture parameterized models of service behavior allowing us to perform adaptations in the form of composition, conversion, or substitution and 2) by the direct modification of structural, functional, and nonfunctional parts of the service model. In both cases, the selection of services and their parts is resolved at runtime via the use of ontological descriptions and semantic matchmaking. Semantic matchmaking has the advantage of increasing the level of detail that it is possible to include in the process of finding matching services, thereby increasing the possibility of finding alternative or more appropriate services.

For the purpose of this paper, our approach was evaluated by application to an industry case study, described as examples in each section.

APPENDIX

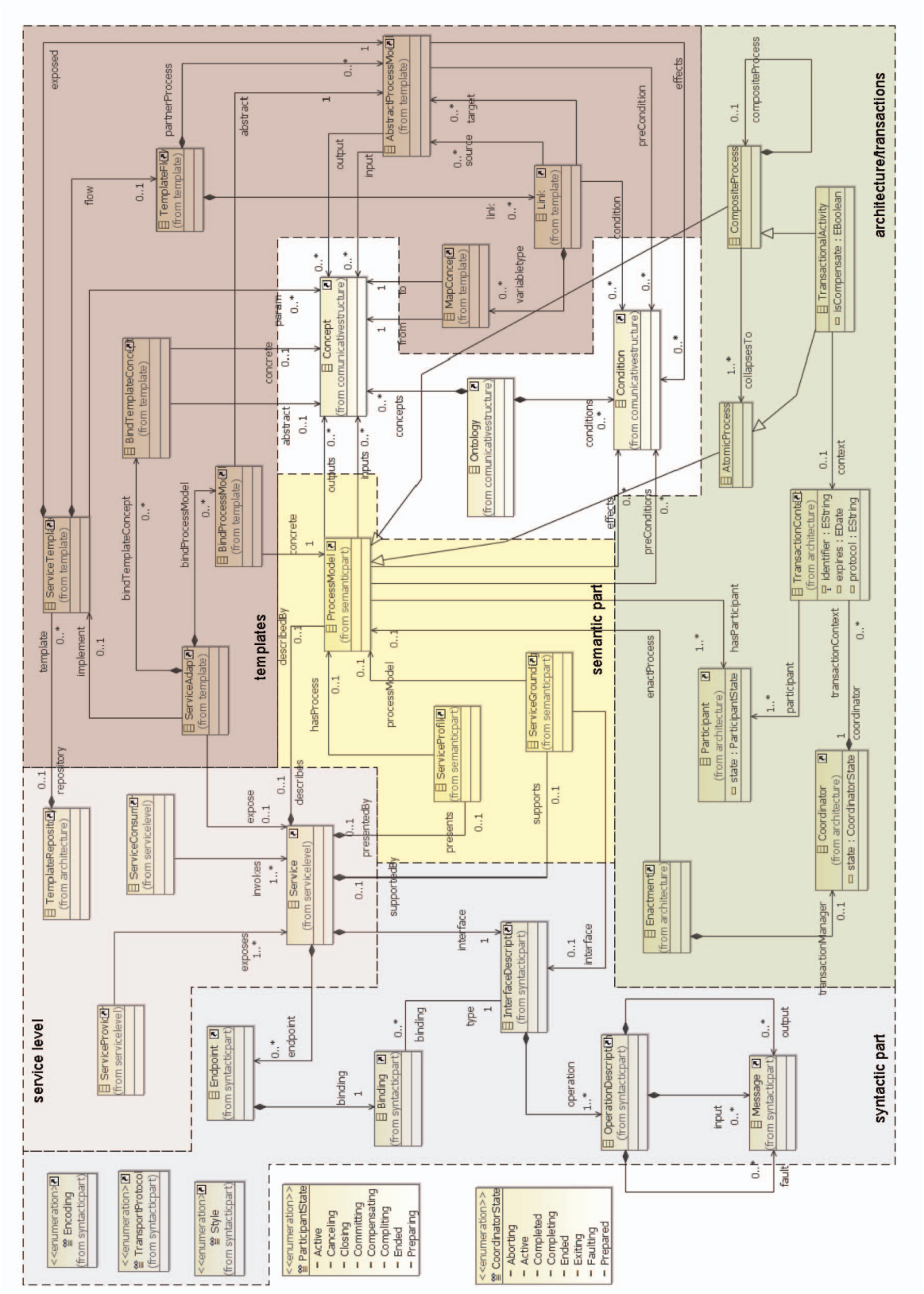

Fig. 14. Service-oriented metamodel (including syntax, semantics, and templates).

ACKNOWLEDGMENTS

This work was carried out as part of the ALIVE project (www.ist-alive.eu), funded by the EU Commission under the 7th framework program. This work was also supported in part by Science Foundation of Ireland grant 03/CE2/I303_1 to Lero—the Irish Software Engineering Research Centre (www.lero.ie).

REFERENCES

[1] M. Dumas, M. Spork, and K. Wang, "Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation," *Business Process Management*, pp. 65-80, Springer, 2006.

[2] A. Brogi and R. Popescu, "Automated Generation of BPEL Adapters," *Proc. Int'l Conf. Service-Oriented Computing (ICSOC '06)*, pp. 27-39, 2006.

[3] J. Harney and P. Doshi, "Speeding up Adaptation of Web Service Compositions Using Expiration Times," *Proc. 16th Int'l Conf. World Wide Web*, pp. 1023-1032, 2007.

[4] N.C. Narendra, K. Ponnalagu, J. Krishnamurthy, and R. Ramkumar, "Run-Time Adaptation of Non-Functional Properties of Composite Web Services Using Aspect-Oriented Programming," *Proc. Int'l Conf. Service-Oriented Computing (ICSOC '07)*, pp. 546-557, 2007.

[5] A. Erradi, P. Maheshwari, and S. Padmanabhuni, "Towards a Policy-Driven Framework for Adaptive Web Services Composition," *Proc. Int'l Conf. Next Generation Web Services Practices*, pp. 33-38, 2005.

[6] "PO-JRA-1.2.1: State of the Art Report, Gap Analysis of Knowledge on Principles, Techniques and Methodologies for Monitoring and Adaptation of SBAs," report, S-Cube, 2008.

[7] S.H. Chang, H.J. La, and S.D. Kim, "A Comprehensive Approach to Service Adaptation," *Proc. IEEE Int'l Conf. Service-Oriented Computing and Applications*, 2007.

[8] B. Selic, "The Pragmatics of Model-Driven Development," *IEEE Software*, vol. 20, no. 5, pp. 19-25, Sept. 2003.

[9] "Coordination, Organisation and Model Driven Approaches for Dynamic, Flexible, Robust Software and Services Engineering," European Commission Framework 7 ICT Project, ALIVE, http://www.ist-alive.eu, 2009

[10] S. Clarke, A. Staikopoulos, S. Saudrais, J. Vázquez-Salceda, V. Dignum, W. Vasconscelos, J. Padget, T. Quillinan, L. Ceccaroni, and C. Reed, "ALIVE: A Model Driven Approach to Coordination and Organisation for Dynamic Services Engineering," *Proc. MODELS 2008 Research Projects Symp.*, 2008.

[11] "OWL-S: Semantic Markup for Web Services," W3C Member Submission, http://www.w3.org/Submission/OWL-S, Nov. 2004.

[12] M.P. Papazoglou and J.-J. Dubray, "A Servey of Web Service Technologies," Technical Report DIT-04-058, Informatica e Telecomunicazioni, Univ. of Trento, 2004.

[13] "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language," W3C Recommendation, http://www.w3.org/TR/wsdl20, June 2007.

[14] "Universal Description, Discovery and Integration (UDDI)," OASIS, http://www.oasis-open.org/specs, 2002.

[15] G. Antoniou and F. vanHarmelen, *A Semantic Web Primer*. MIT Press, 2004.

[16] T. Gruber, "Ontology," *Encyclopedia of Database Systems*, L. Liu and M. Tamer Özsu, eds. Springer-Verlag, 2009.

[17] M. Klusch, B. Fries, and K. Sycara, "Automated Semantic Web Service Discovery with OWLS-MX," *Proc. Fifth Int'l Conf. Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2006.

[18] C. Atkinson and T. Kuhne, "Model-Driven Development: A Metamodelling Foundation," *IEEE Software*, vol. 20, no. 5, pp. 36-41, Sept. 2003.

[19] "OWL Web Ontology Language Overview," W3C Recommendation, http://www.w3.org/TR/2004/REC-owl-features-20040210, Feb. 2004.

[20] "SWRL: A Semantic Web Rule Language Combining OWL and RuleML," W3C Member Submission, http://www.w3.org/Submission/SWRL, 2004.

[21] D. Recordon and D. Reed, "OpenID 2.0: A Platform for User-Centric Identity Management," *Proc. Second ACM Workshop Digital Identity Management*, 2006.

[22] A. Staikopoulos, S. Saudrais, S. Clarke, J. Padget, O. Cliffe, and M.D. Vos, "Mutual Dynamic Adaptation of Models and Service Enactment in ALIVE," *Proc. Models@Runtime Workshop (MODELS '08)*, 2008.

[23] "Eclipse Modeling Framework Project (EMF)," Eclipse, http://www. eclipse.org/emf, 2009.

[24] M. Klusch, B. Fries, and K. Sycara, "OWLS-MX: A Hybrid Semantic Web Service Matchmaker for OWL-S Services," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 2, pp. 121-133, 2009.

[25] "M2M/Operational QVT Language (QVTO)," Eclipse, http://wiki.eclipse.org/M2M/Operational_QVT Language_(QVTO), 2009.

[26] "Web Services Business Process Execution Language (WS-BPEL) Version 2.0," OASIS, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, 2007.

[27] "Graphical Modeling Framework (GMF)," Eclipse, http://www.eclipse.org/gmf, 2009.

[28] S. Katia, S. Widoff, M. Klusch, and L. Jianguo, "LARKS: Dynamic Matchmaking among Heterogeneous Software Agents in Cyberspace," *Autonomous Agents and Multi-Agent Systems*, vol. 5, pp. 173-203, 2002.

[29] S. van Splunter, F.M.T. Brazier, J. Padget, and O. Rana, "Dynamic Service Reconfiguration and Enactment Using an Open Matching Architecure," *Proc. Int'l Conf. Agents and Artificial Intelligence*, 2009.

[30] "Object Constraint Language Specification, Version 2.0," OMG, http://www.omg.org/cgi-bin/doc?formal/2006-05-01, 2006.

[31] P. Maes, "Concepts and Experiments in Computational Reflection," *ACM SIGPLAN Notices*, vol. 22, no. 12, pp. 147-155, 1987.

[32] C. Baral, *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.

[33] F. Fleurey, V. Dehlen, N. Bencomo, B. Morin, and J.-M. Jézéquel, "Modeling and Validating Dynamic Adaptation," *Proc. Models@Runtime Workshop (MODELS '08)*, 2008.

[34] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure," *Computer*, vol. 37, no. 10, pp. 46-54, Oct. 2004.

[35] J. Floch, S. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjorven, "Using Architecture Models for Runtime Adaptability," *IEEE Software*, vol. 23, no. 2, pp. 62-70, Mar./Apr. 2006.

[36] Y. Sam, O. Boucelma, and M.-S. Hacid, "Web Services Customization: A Composition-Based Approach," *Proc. Sixth Int'l Conf. Web Eng.*, 2006.

[37] J. Jiang, A. Ruokonen, and T. Systa, "Pattern-Based Variability Management in Web Service Development," *Proc. Third European Conf. Web Services*, 2005.

[38] K. Birman, R. van Renesse, and W. Vogels, "Adding High Availability and Autonomic Behavior to Web Services," *Proc. 26th Int'l Conf. Software Eng. (ICSE '04)*, pp. 17-26, 2004.

[39] Q.A. Liang, H. Lam, L. Narupiyakul, and P.C.K. Hung, "A Rule-Based Approach for Availability of Web Service," *Proc. Int'l Conf. Web Services (ICWS '08)*, pp. 153-160, 2008.

[40] Q.Z. Sheng, B. Benatallah, Z. Maamar, and A.H.H. Ngu, "Configurable Composition and Adaptive Provisioning of Web Services," *IEEE Trans. Services Computing*, vol. 2, no. 1, pp. 34-49, Jan. 2009.

[41] G. Murphy and C. Schwanninger, "Guest Editors' Introduction: Aspect-Oriented Programming," *IEEE Software*, special issue on aspect-oriented programming, vol. 23, no. 1, pp. 20-23, Jan./Feb. 2006.

[42] R. Hirschfeld and K. Kawamura, "Dynamic Service Adaptation," *Proc. Distributed Computing Systems Workshops*, pp. 290-297, 2004.

[43] W. Kongdenfha, R. Saint-Paul, B. Benatallah, and F. Casati, "An Aspect-Oriented Framework for Service Adaptation," *Proc. Int'l Conf. Service-Oriented Computing (ICSOC '06)*, vol. 4294, pp. 15-26, 2006.

[44] A. Hibner and K. Zielinski, "Semantic-Based Dynamic Service Composition and Adaptation," *Proc. IEEE Congress Services*, pp. 213-220, 2007.

[45] M. Stollberg, D. Roman, J.D. Bruijn, D. Fensel, H. Lausen, A. Polleres, and J. Domingue, *Enabling Semantic Web Services: The Web Service Modelling Ontology*. Springer, 2006.

[46] V.R. Lesser, "Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework," *Proc. First Int'l Joint Conf. Autonomous Agents and Multiagent Systems*, 2002.

Athanasios Staikopoulos received the advanced MSc and PhD degrees in computer science from the University of Birmingham. He is a research fellow at Trinity College Dublin. His research interests include model-driven engineering, component-based development, and coordination and composition of (Web) services.

Owen Cliffe received the BSc and PhD degrees in computer science from the University of Southampton and the University of Bath, respectively. He is a research officer at the University of Bath. His research interests include nonmonotonic reasoning, knowledge representation, declarative problem solving and representing, and reasoning about normative and legal aspects of distributed and autonomous systems.

Razvan Popescu received the BSc and PhD degrees in computer science from the "Politehnica" University of Bucharest and the University of Pisa, respectively. He is a research fellow at Trinity College Dublin. His research interests include the discovery, composition, and adaptation of (Web) services.

Julian Padget received the BSc degree from the University of Leeds and the PhD degree in computer science from the University of Bath. He is a senior lecturer at the University of Bath, where he leads the Agents Research Group. His research interests range from intelligent agents, electronic commerce, and distributed systems to mathematical Web services, programming language design, and computer music.

Siobhán Clarke received the BSc and PhD degrees in computer science from Dublin City University. She is a senior lecturer and fellow at Trinity College Dublin, where she leads the Distributed Systems Group and is a research area leader at Lero: The Irish Software Engineering Research Centre. Her research interests include design and programming models for advanced distributed systems.