Use-case driven service modelling with XML-based tailoring for SOA

Yukyong Kim and Kyung-Goo Doh*
Department of Computer Science and Engineering, Hanyang University ERICA, Ansan 426-791, Korea
Email: yukyong@hanyang.ac.kr
Email: doh@hanyang.ac.kr
*Corresponding author

Abstract: A key problem in service-oriented computing is how to extract business requirements and build solutions with available services. The manual process of eliciting business requirements from use cases and scenarios is time consuming. This paper proposes a semi-automatic way of deriving service models from business use cases using XML-based use-case tailoring. By matching the required capability of each use case to the available capabilities provided by services, the optimal set of use cases is realised by the appropriate set of services. We rewrite use cases in XML to achieve the clear separation of any semantic parts of use-case descriptions from visual representations. The resulting use cases are subsequently tailored into well-defined business functionalities, which are built as services according to service orientation principles. Finally, we analyse and discuss validation results of a case study to illustrate how the proposed approach works.

Keywords: service-oriented architecture; service extraction; use-case model; XML description.

Reference to this paper should be made as follows: Kim, Y. and Doh, K.-G. (2013) ‘Use-case driven service modelling with XML-based tailoring for SOA’, Int. J. Web and Grid Services, Vol. 9, No. 1, pp.35–53.

Biographical notes: Yukyong Kim is a Research Professor in the Department of Computer Science and Engineering at Hanyang University ERICA, Korea. She received a PhD in Computer Science from Sookmyung Women’s University, Seoul, Korea. She was a post-doctoral researcher in the Department of Computer Science at the University of California at Davis, CA, USA. Her research interests include service-oriented architecture, semantic web, quality of services and software quality assurance.

Kyung-Goo Doh is a Professor in the Department of Computer Science and Engineering, Hanyang University ERICA, Korea. He received a PhD in Computer Science from Kansas State University, USA. His research interests are programming languages, program analysis, software security and software engineering.
1 Introduction

1.1 Background

Service-Oriented Architecture (SOA) promotes the idea of assembling application components into a set of services that are loosely coupled to create flexible, dynamic business processes and agile applications (Quintero et al., 2010). In particular, SOA as an architectural style aims to provide modular enterprise business solutions that can extend or change on demand. Because SOA solutions are composed of reusable services, with well-defined, open and published interfaces, SOA also provides a mechanism for integrating existing legacy applications, regardless of their platform or language.

Although the most effective approaches for achieving the full benefits of SOA have been intensely debated in recent years, solutions for the wider business and architectural issues involved in designing high-quality SOA systems for discrete enterprises are still in early stages (Blanco et al., 2012).

In the development lifecycle of SOA, most projects undergo a modelling phase in which critical service components are defined based on requirements that emerge during domain analysis. This paper is motivated by the need for service modelling methodologies to identify and specify the assets of candidates for SOA. A particular type of problem commonly occurs in the modelling phase of deriving services from the classes or components of projects, because existing approaches to programming rely on object-oriented (OO) methods or Component-Based Development (CBD). Although it can be argued that OO analysis and design techniques are good starting points for services, objects are abstractions on a micro-level, whereas services are business-aligned entities. As such, services are at a much higher level of abstraction than objects and components. Accordingly, service modelling needs an appropriate approach to identifying and defining services as an essential starting point for wider application.

On a business level, service orientation enables organisations to expose and offer operations as business services to business partners in order to facilitate on-demand collaboration opportunities. A business service is a specific set of actions that are performed by an organisation (Sanz et al., 2007). Since the operations of an organisation can be analysed on different levels of granularity, business services can represent these operations on different levels as well (Sehmi and Schwegler, 2006). Business services are aligned with the capabilities of an organisation in order to reflect the individual operations. As shown in Figure 1, a business service is realised by one or more application service. A business process describes the order of required tasks that must be accomplished in order to achieve a desired outcome for overall productivity and success (Bernal and Morisio, 2010). Because decomposition of the business domain into functional areas comprising the domain provides high-level business use cases, use cases can be good candidates for business services. The business use-case model provides a software-independent description of business processes. The details associated with business use cases are documented in business use-case specifications. Application services have found their most promising incarnation in the form of web services as the technological implementation of SOA (Kohlborn and Korthaus, 2009). While technology and standards are important in building SOA applications; however, they are not sufficient on their own. Moving to service orientation is a non-trivial endeavour, which requires far more than simply wrapping software entities in web service interfaces.
1.2 A research question

Even though a wide range of advantages associated with the introduction of SOA are currently advocated, an effective service modelling approach for achieving the full benefits of SOA has not yet been reached. Comprehensive and detailed approaches for identifying and designing services must occur in the earliest phases of SOA adoption. Detailed modelling guidelines that lead to prescriptive modelling techniques are necessary to answer a critical question:

‘How do we derive “good” service abstractions from high-level business requirements and business models?’

The objective of the current research is to propose a method for identifying and specifying services as elements of SOA at proper abstraction levels based on business use cases. In our previous work (Kim and Doh, 2007), refactoring-based extraction method is defined. As an extension of the work, the XML-based approach for making semi-automated decisions that we propose in this paper is expected to be efficient for identifying services. In this paper, representative services are derived from business use cases by tailoring. Tailoring is a process to select the correct levels of abstraction and granularity, to identify commonalities, and to eliminate redundancies among use cases. As a cohesive, end-to-end service delivery methodology, the method proposed in this study facilitates the easy definition of coarse-grained candidates for service. Using eXtensible Markup Language (XML), we formally refine use cases by achieving the clear separation of the semantics of use cases from their visual representations.

This paper is organised as follows. After starting with a brief introduction to existing research about service modelling in Section 2, Section 3 describes the service modelling process, including definitions of terms and basic principles used in the paper. Section 4 presents rules for tailoring use cases, and Section 5 defines construction of the proposed service model. A case study is utilised to validate our method in Section 6. Finally, Section 7 summarises and concludes the paper with suggestions for future works.
2 State of the art

Although there is currently no explicit step or method in existing methodologies to identify and define services, as acknowledged by Shirazi et al. (2009) and Gu and Lago (2010), a number of methodologies have been proposed for service modelling approaches to identify and specify services. Most of the existing methodologies address the huge demand for process guidance in SOA projects.

Jain et al. (2004) describe a formal approach to web services identification, using an analysis-level object model as input. The study generates potential web service designs, in which classes in the object model are grouped into appropriate web services based on static and dynamic relationships between classes, and subsequently employs a maximum spanning tree algorithm. Erl (2005) suggests a service-oriented analysis and design methodology. This methodology is a step-by-step guide through the two main phases of analysis and design. Analysis activities take a top-down business view in which service candidates are identified. Service Oriented Modelling and Architecture (SOMA) is a full-blown modelling methodology by IBM consisting of three steps. The steps are identification, specification and realisation of services, flows (business processes), and components realising services (Arsanjani, 2005). The process is highly iterative and incremental. Because SOMA is proprietary to IBM, however, its full specification is not publicly available. The concept of Web Services Modeling Framework (WSMF) has been defined to provide a rich conceptual model for the development and description of web services (Fensel et al., 2002). The philosophy of WSMF is based on the maximal decoupling principle. A model in WSMF consists of the four main elements of ontology, goal repositories, web services descriptions and mediators, which bypass interoperability problems. A method known as Service-Oriented Unified Process (SOUP), developed by Mittal (2006), is primarily based on the Rational-Unified Process (RUP) method. The SOUP lifecycle consists of six phases of programming, which are: incept, define, design, construct, deploy and support. SOUP lacks detailed documentation and leaves room for adaptation (Mittal, 2006). The method is used in two slightly different variations, one of which adopts RUP for initial SOA projects, and another that utilises a mix of RUP and XP for the maintenance of existing SOA rollouts. Bell (2008) defines the Service-Oriented Modelling Framework (SOMF). In SOMF, the conceptual service identification process is based on six best practices that assist with identifying organisational concepts, establishing conceptual services, founding service associations, and forming service structures. To discover abstractions and derive conceptual services, the method utilises a decision tree. The attribution analysis process yields sets of recommended attributes that are essential inputs for most categorisation activities. Conceptual services are derived from the attributes through the application of corresponding business. Surveys of service-oriented development methodologies have been found in the work of Ramollari et al. (2007), one of which (Hubbers et al., 2008) provides a high-level overview of ten frequently used approaches for identifying services. Most surveys conclude that no one method of service-oriented computing is perfect, but rather that each method has certain benefits and trade-offs.

Optimal service granularity is crucial for ensuring maximum reuse in SOA. Typically a balance is established depending on the levels of abstraction, likelihood of change, complexity of services, and the desired level of cohesion and coupling. Current approaches, however, are far from accurately accounting for the proper levels of granularity of services. Moreover, because current approaches provide only descriptive
guidelines to define services, application of these approaches is not straightforward and objective, but instead is reliant on the relative experience and intuition of developers. The main goal of this paper is to provide a conceptual framework for identifying services in a non-arbitrary fashion. Our work intends to define a process for identifying services at the right levels of abstraction through a formal and systematic approach based on XML.

3 Overview

In SOA environments, a service is typically published and requested by different participants as a black box. Finding a business solution involves aligning IT practices with business goals. In order to realise a business goal in terms of service-oriented computing, various heterogeneous black box services from local or remote providers must collaborate harmoniously in a way that is characterised by loose coupling and reuse (Erl, 2007). This paper defines services as follows:

- Conceptual service is a service that is not yet implemented (implementation agnostic), which may or may not be a software service.
- A conceptual service identified during an analysis becomes a service candidate for design.
- A service operation candidate is a service operation identified from functional requirements. It might become a service candidate itself, or it might comprise (i.e. be aggregated in) a (more complex) service candidate.
- Business service is (ideally) a self-contained, stateless business function that accepts one or more requests, and returns one or more responses, through a well-defined, standard interface. During service identification, the elicited business services become the service candidates of the design phase. They might be service operation candidates.

Service modelling is the process of determining how business requirements can be represented through IT services. Service modelling consists of two steps: (a) identifying service candidates and service operation candidates and (b) constructing the business services model. In this paper, service identification starts with business use cases partitioned from business functionalities. A business use case facilitates description of what the system will do at high levels to capture overall business processes. These business use cases are often very good candidates for exposing business services at the edges of enterprises, or for internal business services utilised across lines of business (Arsanjani, 2005). Accordingly, defining services by extracting coarse-grained services from business use cases may be more efficient than existing methods for defining services.

In order to separate the semantic parts of use cases from visual representations, and to formally model the existing elements of a domain, we use XML for encoding textual use-case descriptions in a machine-readable form. This process establishes a common vocabulary to eliminate ambiguities between different stakeholders. It also allows for the formal reasoning of concepts and instances to discover new relationships or detect inconstancies.
As shown in Figure 2, our approach to modelling business services includes three activities. We first rewrite use cases in XML to formally and unambiguously apply tailoring. In business use-case models, it is determined for each use case whether or not the use case is supported by the application to be built. To alleviate a service proliferation syndrome in which an increasing number of small-grained services are defined, designed and deployed with very little governance, the results of which are major performance, scalability, and management issues use cases should be refined and composed of finer-grained use cases. By defining a meta-model for use cases in Document Type Definition (DTD), we can rewrite use cases in XML, and subsequently, validate them.

**Figure 2** Execution view

Once generated in XML, use cases are refined by tailoring. We define tailoring as the reorganisation of use cases to select the correct levels of abstraction and granularity, identification of commonalities, elimination of redundancies, and reuse of functionality. The input for this phase is business use-case models and a set of business terms.

Service candidates are refined by iterative tailoring. Finally, we identify business services from the candidates by defining interfaces of each service in XML-based descriptions such as Web Service Definition Language (WSDL). Services expose their functionality by using Web Service Description Language (WSDL; Mateos et al., 2011). There are five basic parts in WSDL 2.0, including Description, Types, Interfaces, Binding, and Service. The `<Description>` element describes the target name space as a root tag. `<Types>` and `<Interfaces>` define SOAP messages in a platform-independent and language-independent manner, as the abstract section. The `<Binding>` and `<Service>` elements bind the interface to transport and the binding to an end point, respectively. Site-specific matters such as serialisation are relegated to these aspects of WSDL.

We deliver the `<Types>` and `<Interfaces>` elements by defining interfaces. `<Types>` refers to machine-independent and language-independent type definitions. The element of `<Interfaces>` includes operation, input parameters and output parameters. `<Interfaces>` also contains `<fault>` definitions, as well as `<operation>`. The `<input>` and `<output>` tags both refer to the `<type>` aspect of WSDL.
4 Tailoring of use cases

This section first describes a meta-model for extended use cases based on the Unified Modelling Language (UML) meta-model. To refine use cases, we analyse steps in use cases and apply tailoring rules to use cases. We present a method to analyse steps by tagging each step, and subsequently, we present a set of rules for tailoring.

4.1 Meta-model for extended use cases

A use case is the specification of a sequence of actions, including variants that a system can perform by interacting with actors of the system (OMG, 2010). Because the UML defines use cases at an abstract level by providing only an external view of use cases, the definition of how concrete behaviour is specified for any use case is left open. Moreover, the description form of use cases is mostly natural language text, unlike formalisms such as interactions, activities and state machines. This paper selects the format specified by Cockburn (2001) as our preferred template for use cases. This particular use-case format is an example of a structured use case format (Cockburn, 2001). Table 1 shows an example of the template specified by Cockburn. In the main success scenario and extensions, each step is a statement describing an activity between actors in a system (primary actor, system and secondary actor).

Table 1 Example of the preferred template for use cases (Cockburn, 2001)

<table>
<thead>
<tr>
<th>UC # 1010: Buying stocks on the web</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Primary Actor:</strong> Purchaser/user</td>
</tr>
<tr>
<td><strong>Scope:</strong> Personal Advisors/Finance package (PAF)</td>
</tr>
<tr>
<td><strong>Level:</strong> Task</td>
</tr>
<tr>
<td><strong>Precondition:</strong> User already has open PAF.</td>
</tr>
<tr>
<td><strong>Successful End Condition:</strong> Remote website acknowledges the purchase of stocks. Both PAF logs and user’s portfolio are updated.</td>
</tr>
<tr>
<td><strong>Failed End Protection:</strong> Sufficient log information exists that PAF detects that something has gone wrong. User is asked to provide the details.</td>
</tr>
<tr>
<td><strong>Main success scenario:</strong></td>
</tr>
<tr>
<td>1. User selects stocks to buy on the web.</td>
</tr>
<tr>
<td>2. PAF obtains the name of website to use for purchase (E*Trade, Schwabb, etc.)</td>
</tr>
<tr>
<td>3. PAF opens web connection to the site and retains control.</td>
</tr>
<tr>
<td>4. User browses and buys stock from the website.</td>
</tr>
<tr>
<td>5. PAF intercepts responses from the website and updates the user’s portfolio.</td>
</tr>
<tr>
<td>6. PAF shows the user the standing of the new portfolio.</td>
</tr>
<tr>
<td><strong>Extensions:</strong></td>
</tr>
<tr>
<td>2a. User wants to purchase stocks from a website that PAF does not support:</td>
</tr>
<tr>
<td>2a1. System gets new suggestion from user, with option to cancel use case.</td>
</tr>
<tr>
<td>4b. Website does not acknowledge purchase, but rather puts the purchase on delay:</td>
</tr>
<tr>
<td>4b1. PAF logs the delay and sets a timer to ask user about the outcome.</td>
</tr>
<tr>
<td>4b2. (See use case update questioned purchase).</td>
</tr>
</tbody>
</table>
In order to semi-automatise the transformation process and ensure consistent tailoring, we define a meta-model for use cases. Existing UML use-case models either contain too little information for our purposes, or do not provide appropriate design details. Formalising a meta-model connection ensures that our proposed approach is applicable to Model-Driven Development (MDD). Figure 3 shows a meta-model for extended use cases. Classes with a white background are imported from the UML meta-model, and accordingly, are related to the identical classes presented by OMG (2010). Classes with the filled background are intended to define tailoring in use cases. A use-case model consists of actors and use cases, as well as the relationships between them. Any use case includes a main success scenario that represents the main sequence of steps, which captures the use case behaviour.

Figure 3  Meta-model for extended use cases (see online version for colours)

4.2 Structuring use cases in DTD

The requirements of systems are generally specified by textual use cases due to the concrete, narrative style of use-case expression, which is very effective for eliciting user requirements. In order to automatically obtain required attributes, all kinds of activity data need to be extracted from each step of a use case. In order to accomplish this, we define use-case structure with DTD. By translating the meta-model, we create a DTD description for XML documents of use-case models, as shown in Table 2.

In textual use-case descriptions, one step describes only one activity of the actor, even though the sentences are not stylistically well defined. In order to rewrite use-case descriptions into XML for steps that contain no abort or goto, the first verb related to the actor, and the first noun after the verb as an object, together form the step. When the step does contain abort or goto, the activity is specified by abort or goto# (# is a step number), respectively. Consider the example use case shown in Table 1, the description of which is written as an UCMModel in XML as shown in Table 3.
Use-case driven service modelling with XML-based tailoring

4.3 Rules of tailoring use cases

The goal of tailoring use cases is the selection of the correct levels of abstraction and granularity, identification of commonalities, elimination of redundancies, and reuse of
functionality to identify service candidates. Our approach preserves the semantics of use-case models because we analyse and subsequently realign the steps of a use case without creating or deleting steps. To find common steps and to eliminate redundancies caused by analogous business terms, we assume that domain-specific terms, as well as general-purpose terms usually adopted in a given system, are organised according to semantic relationships such as synonymy.

We can implement this tag comparison using existing string comparison algorithms, or sequence alignment algorithms. Based on the above definition, we define two rules of tailoring for use cases.

4.3.1 Decomposition

The decomposition rule involves partitioning a use case into more cohesive use cases with the right levels of abstraction. When common steps exist for several use cases, a new use case is created from those common steps. They are connected by an `<include>` relationship, as shown in Figure 4.

Figure 4  Decomposition (see online version for colours)

- Rule 1 (Decomposition): For \( uca = \{ s_j, \ldots, s_k \} \) and \( ucb = \{ s_l, \ldots, s_m \} \), if there exists a sequence of steps \( \{ s_p, \ldots, s_q \} \subseteq \{ s_j, \ldots, s_k \} \) and \( \{ s_p, \ldots, s_q \} \subseteq \{ s_l, \ldots, s_m \} \), then decomposing \( uca \) and \( ucb \) according to the sequence of steps produces a new use case \( ucc = \{ s_p, \ldots, s_q \} \) with \( uca' \) and \( ucb' \) standing for \( uca' = \{ s_j, \ldots, s_k \} - \{ s_p, \ldots, s_q \} \) and \( ucb' = \{ s_l, \ldots, s_m \} - \{ s_p, \ldots, s_q \} \), respectively. Then the use cases \( uca' \) and \( ucb' \) include the use case \( ucc \).

Because too many fine-grained services may result in high-message traffic between service users and providers, great care must be taken in applying the composition rule to use cases. Cognitive science provides a good solution in the magical number seven, plus or minus two, widely known as an indicator of complexity. A use case consists of five steps or less will be too fine-grained, and thus it is generally not to be decomposed.

4.3.2 Composition

In contrast, composition refers to combining several use cases into more coarse-grained use cases. Regarding the composition of use cases, this paper deals with extend and inheritance relationships.

When there is an `<extend>` relationship between use cases, the level of granularity can be regulated by combining the two related use cases into a new use case. The extend relationship specifies that the incorporation of the extension use case is dependent on
what happens when the base use case executes. That is, the extension use case owns the extend relationship. The extension use case can specify several extend relationships for a single base use case. While the base use case is defined independently and is meaningful by itself, the extension use case is not meaningful on its own. The extension use case consists of one or several steps, which describe additional behaviours that incrementally augment the behaviour of the base use case. Each step can be inserted into the base use case at a different point, called an extension point. Within this context, extend relationships among use cases should be reconstructed in terms of SOA. The level of granularity can be regulated by combining extension use cases into their base use case.

Figure 5 Composition on extension (see online version for colours)

- **Rule 2 (Composition on extension):** For \( uca \) = \{s_p, \ldots, s_q\} and \( ucb \) = \{s_j, \ldots, s_k\}, when the use case \( ucb \) extends the use case \( uca \) (i.e. there exists an extend relationship from \( ucb \) to \( uca \)), we combine \( ucb \) into \( uca \) to be \( uca = \{s_p, \ldots, s_q\} \cup \{s_j, \ldots, s_k\} \).

The use-case description has an extension point to represent the \( \text{extend} \) relationship, as shown in Table 1. To perform composition on the extend relationship, we can append the steps of extension use cases in each statement of the extensions part of the description. Accordingly, the composed part is rewritten by the \( \text{extension} \) tag in XML.

Inheritance between use-cases are generalisation/specialisation relationships. More specifically, because UML is a standardised general-purpose modelling language in the field of object-oriented software engineering, there exists the concept of inheritance among use cases as a type of generalisation/specialisation relationship. Tightly coupled architecture such as inheritance is not appropriate for SOA, which is basically composed of loosely coupled services. From this perspective, generalisation/specialisation relationships among use cases should be reconstructed according to principles of SOA. Inheritance is a way to reuse existing objects. Thus through inheritance, use cases at low levels of the hierarchy have finer granularity than use cases at high levels. By combining derived use cases into a base use case, the level of granularity can be regulated.

- **Rule 3 (composition on inheritance):** For \( uca \) = \{s_p, \ldots, s_q\} and \( ucc \) = \{s_j, \ldots, s_k\}, when the use case \( uca \) is derived from the use case \( ucc \) (i.e. \( uca \) is \( ucc \) with additional steps), we combine \( uca \) into \( ucc \) to be \( ucc = \{s_j, \ldots, s_k\} \cup \{s_p, \ldots, s_q\} \).

Derived use cases are composed into a base use case by describing IF-THEN statements in steps. The relationship is considered as alternative flows according to the conditions of each step.
Tailoring use cases refers to identifying correctly grained services by defining use cases with the right levels of abstraction and granularity through the rearrangement of action steps without making any changes. A fine-grained service has low reusability and low flexibility of service functionalities. Moreover, because many fine-grained services cause heavy network traffic between service providers and service consumers, it is necessary to build services with proper granularity by means of division and merging.

5 Transitioning from use cases to business services

This section defines business services from tailored use-case models and subsequently presents how to build service descriptions. Because service models are abstractions of the IT services of enterprises, any service model is a comprehensive, composite work-product that encompasses all services, providers, specifications, partitions, messages and collaborations, as well as the relationships among these elements.

Rational SOMA defines the key aspects of service models as service identification, service specification, and service realisation decision. A service model is needed to identify candidate services (i.e. capabilities that ‘might’ be implemented as exposed services) and to capture decisions about which services actually will be exposed, to specify the terms of contracts between service providers and service consumers, and to associate services with the components required to realise these services (Dunnavant and Johnston, 2011).

Within a more coarse-grained perspective on service capability, this paper derives XML descriptions of service candidates and service hierarchy by identifying and capturing decisions about which services to expose. We derive service candidates from tailored use cases, and subsequently, service hierarchy from the \langle include\rangle relationships between use cases.

In defining a service description in this paper, a business service candidate is denoted by \langle sName; op_1, \ldots, op_n\rangle, where sName is a service name, and each op is a functional sequence based on the who property in a pair of tags, \langle step\rangle and \langle/step\rangle. For example, the use case of ‘Buy stocks over the web’ in Table 3 has op_1 as the user and op_2 as the PAF, as shown in Table 4. As a flow of events of a use case in the design model, functional sequences of each use case are interfaces of the service that realises the use case. Functional sequences are built by connecting all the activities in the use case with expressions. Three expressions are used, as follows.
• Sequential execution $\text{step}_1; \text{step}_2; \ldots; \text{step}_n$ – In the main scenario of a use case, the basic flow scheme is sequential in that if $\text{step}_i$ is followed by $\text{step}_{i+1}$, then an activity corresponding to $\text{step}_{i+1}$ would follow the successful completion of an activity corresponding to $\text{step}_i$.

• Alternative execution $\text{If (cond: step}_1; \text{step}_2)\text{ – Execution of steps is conditional by if-then statements. Alternative execution is used for the activity of step}_2\text{ from the main scenario, and alternated activity of step}_1\text{ in extensions.}$

• Call execution $\text{Call(uName)}$ – If a step has an ‘include’ property, then the step is denoted by a call statement in which a parameter $uName$ is the name of the use case, according to the name of the service achieved.

Table 4  Business service definitions extracted from Table 3

<table>
<thead>
<tr>
<th>Service: BusinessService</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Service: BusinessService&gt;</td>
</tr>
<tr>
<td>&lt;sName = Buy stocks over the web&gt;</td>
</tr>
<tr>
<td>&lt;op who=User&gt;</td>
</tr>
<tr>
<td>selects stocks;</td>
</tr>
<tr>
<td>browses a stock;</td>
</tr>
<tr>
<td>$\text{If (Web site does not acknowledge: op-PAF (logs the delay; sends a timer; \text{call (Update purchase)}); buys a stock);}$</td>
</tr>
<tr>
<td>&lt;/op&gt;</td>
</tr>
<tr>
<td>&lt;op who=PAF&gt;</td>
</tr>
<tr>
<td>$\text{If (PAF does not support; gets new suggestion; gets name);}$</td>
</tr>
<tr>
<td>opens connection;</td>
</tr>
<tr>
<td>intercepts responses;</td>
</tr>
<tr>
<td>updates users portfolio;</td>
</tr>
<tr>
<td>shows the new portfolio;</td>
</tr>
<tr>
<td>&lt;/op&gt;</td>
</tr>
<tr>
<td>&lt;/Service: BusinessService&gt;</td>
</tr>
</tbody>
</table>

The formal definition is in the form of service WSDL based on XML. We define the business service as an XML-like statement that captures some additional service information. We subsequently provide service descriptions in WSDL or BPEL4WS to define the identified business services. WSDL describes the public interface for the web service. The term <interface> is a WSDL 2.0 term to define a web service, the operations performed, and the messages that are used to perform the operation. Each statement in an <op> tag is surrounded by an <operation> tag. Operations with who=’User’ are excluded, however, because they are not actual operations in terms of system functionalities. An operation that is described by WSDL has input and output variables, which are mapped to <input> and <output> tags.

Table 4 shows logical descriptions of system components that realise services. In specifying services, we must decide on implementation strategy. Service implementation strategy is about whether the service is implemented, wrapped, or requested. If there is no service to fulfil the requirements of business functionalities, we must define which service candidates are implemented by reuse and/or new development. Through this decision, a set of application services is defined.
6 Evaluation

6.1 Comparison with other approaches

Many approaches have been proposed for service modelling, including SOMA and SOMF. Most of the existing methods start from business processes and enterprise-level information, taking both business needs and context into consideration. This approach is consistent with the fact that SOA design intends to realise software reuse through large-grained services specifically intended to create business value.

One existing study (Gu and Lago, 2010) identifies and reviews 30 primary Service Identification Methods (SIMs). According to the results, most SIMs are based on the business process decomposition strategy. Business process decomposition strategy is suggested as one of the ten most common strategies for identifying services in other existing literature as well in the Hubbers et al.’s (2008) study. We compare our method with five other service modelling methods from the primary SIMs identified in the work of Gu and Lago (2010). The comparison matrix is provided in Table 5.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Output Format</th>
<th>Validation</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wang et al. (2005)</td>
<td>Formal</td>
<td>Evaluated</td>
<td>Algorithm</td>
</tr>
<tr>
<td>Inaganti and Behara (2007)</td>
<td>Informal</td>
<td>–</td>
<td>Guidelines</td>
</tr>
<tr>
<td>Amsden (2007)</td>
<td>Informal</td>
<td>Example</td>
<td>Analysis</td>
</tr>
<tr>
<td>Yousef et al. (2009)</td>
<td>Informal</td>
<td>Case study</td>
<td>Ontology</td>
</tr>
<tr>
<td>Bianchini et al. (2011)</td>
<td>Informal</td>
<td>Example</td>
<td>Algorithm</td>
</tr>
<tr>
<td>Our method</td>
<td>Formal</td>
<td>Case study</td>
<td>Algorithm</td>
</tr>
</tbody>
</table>

Identified services are described in informal service specifications in many approaches. A service model is used to illustrate the relationships among identified services. A service model that utilises formal specification is useful for describing composite services because formal service specifications are often used to describe services identified under more formal techniques. Further, formal specification is relatively explicit in terms of implementing the identified service components.

Among service finding techniques, less formal techniques involve guidelines that provide advice, such as how to identify services from goal-scenario models and how to map tasks from business process models to services. Analysis is a technique that is more abstract, thereby requiring its users to deeply understand the problem they face and to make motivated decisions. Some methods formally codify rules to specify the way in which services are identified, such as with algorithms or ontology.

In comparison to existing methods, our method is intended to complement other approaches by elevating objectivity through the use of XML-based modelling of use cases. We expect our method to be clear, objective, and efficient for identifying services at the proper abstraction levels in order to provide a systematic approach to making semi-automated decisions in business processes.
6.2 A case study

To validate the effectiveness of the proposed method, we show how the approach works for a simple order-to-invoice business system presented in UBL 2.0 (OASIS, 2006). The method extends the order-to-invoice processes to cover a supply chain from sourcing to payment, including the commercial collaborations of international trade. Figure 7 illustrates the context of the process assumed by UBL 2.0 documents. Note that relationships between actors are beyond the scope of this paper, and consequently, are excluded from tailoring.

*Figure 7 Use case diagram (OASIS, 2006) (see online version for colours)*

According to the composition on inheritance Rule 3, the generalisation/specialisation relationships between use cases ‘Catalogue provision’, ‘Fulfillment’, ‘Billing’ and any of their derived use cases, are eliminated. Applying Rule 2 of composition on extension, the use cases ‘Payment’ and ‘Statement of account’, which are connected by an <extend>, are composed into the base use case ‘Payment’. All use cases having <extend> relationships are similarly composed. According to Rule 1, a new use case of ‘Account validation’ is added from the common steps of two use cases ‘Payment’ and ‘Billing’. Figure 8 shows the result of tailoring.

From this tailored use-case model, we redefine an XML document according to DTD. We generate WSDL 2.0 documents for these services as service technology architecture from service component architecture, as shown in Figure 9.
Figure 8  Use case diagram refined by tailoring (see online version for colours)

Figure 9  Service component architecture
7 Conclusions

Most SOA projects undergo a modelling phase in which critical service components are defined from requirements that emerge during domain analysis. Once created, the services may be deployed. However, there is a particular problem in the modelling phase of deriving services from classes or components. Objects are on micro-level abstractions, while services are business-aligned entities, and are therefore at a much higher level of abstraction than objects and components. To resolve this problem, our research demonstrates a phased approach to defining services from use-case models. An effective service modelling approach for achieving the full benefits of SOA has not yet been reached in existing research. This study focuses on how to derive ‘good’ service abstractions from high-level business requirements in terms of UML use-case models, and subsequently defines a comprehensive and detailed approach for identifying and designing services.

In this paper, services are derived from business use cases by tailoring. Tailoring is a process to select the correct levels of abstraction and granularity, identify commonalities, and eliminate redundancies among use cases. As a cohesive, end- to-end service delivery methodology, the proposed method facilitates the rapid definition of coarse-grained service candidates. Using XML, we formally refine use cases by achieving clear separation of the semantics of use cases from their visual representations. Furthermore, we define an appropriate structure for use-case descriptions using DTD. Finally, we present a case study to validate our approach.

Regarding future research, we first plan to extend our work to the formalisation of use cases in order to produce more precise service descriptions, including SLA contractions. Second, because we currently have use-case descriptions from XML generators only, we intend to fully implement and to incorporate our method in a CASE tool. Finally, we intend to integrate our approach with the entire service modelling framework to define service models from source codes as well as business models.

Acknowledgements

This work was supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology (MEST)/National Research Foundation of Korea (NRF) (Grant 2012-0000469).

References


Use-case driven service modelling with XML-based tailoring


