Adaptation of Service-based Business Processes by Context-Aware Replanning

Antonio Bucchiarone, Raman Kazhamiakin, Marco Pistore and Heorhi Raik
Fondazione Bruno Kessler
Via Sommarive, 18, Trento TN 38100, Italy
{bucchiarone,raman,pistore,raik}@fbk.eu

Abstract—Business processes are typically used by organizations to meet a specific business goal by executing a set of coordinated activities realized through Web services and service compositions. Operating in open and dynamic environments, business processes often need to be adapted during the execution to react to changes and unexpected problems. The aim of this paper is to provide a dynamic and flexible way to adapt business processes to run-time context changes that impede the achievement of a business goal. We define a formal framework that uses a planning technique to adapt the execution of the service-based business process at runtime in case of context changes. The adaptation enables the business process to continue its normal execution by recovering it to a context, in which the original goal can be achieved. The proposed solution is implemented and validated using a scenario from the logistics domain.

Keywords—Business Processes, Adaptation, Services, Context, Planning.

I. INTRODUCTION

In recent years, service-oriented architectures have been widely used for the realization of complex business processes [1]. Services and service compositions are exploited in a wide range of application domains, such as logistics, supply chain management, user-centered processes, etc. In such processes the activities are realized through the invocation of a set of available services that could be software services but also human-based services, where the human actors realize particular tasks.

Modern business processes often operate in very dynamic, open, and distributed environments, where the relevant properties are changing continuously. Sometimes these changes can compromise the overall execution of the process causing the failure of the entire application execution. In order to react to such failures, there is a need for mechanisms that bring the process execution back to the context from which the business goals can be achieved and the normal execution can proceed. One way to have a business process adaptable is to consider a priori all the cases in which the process can deviate from the normal behavior (i.e., Exception Handling [2]). In this way, it is possible to completely characterize the reaction of the system at design time as the activities (or even subprocesses) implementing the recovery behavior. This specification may be performed by extending standard languages (e.g., BPEL) with the adaptation-specific tools [3], [4], using a set of predefined adaptation rules [5], [6], using aspect-oriented approaches [7]–[9], or modeling and managing business process variants [10]. However, in many situations, such approaches fail to completely solve the problem of adaptation. First, for complex processes the variety of alternatives that require specific recovery actions may be too large for the designers to consider, making the business process hard to define, implement, and revise. And still, unexpected situations and changes still may occur at run-time, requiring the adaptation to be performed even if the concrete case (and its handling) has not been anticipated at design time. To deal with such situations, the adaptation mechanisms should be much more dynamic and flexible, so that the process can recover from critical deviations without defining them a priori.

In this paper, we aim at providing a formal framework that uses an efficient automated planning technique to adapt the execution of the business process at runtime considering the state of the context. This approach relies on the two key ideas. First, adaptation activities are not explicitly represented and are not associated to a concrete context situation or context change. Instead, they are dynamically derived from the currently observed context, the state of a business process, and business goals. To support this, we propose an explicit yet compact and efficient model of context, services, and business goals. Second, we propose an execution environment, where the context of the process is continuously observed, and the adaptation is triggered in case of deviations from what is expected by the business process activities and policies.

The rest of the paper is structured in the following way. In Section II we present our motivating example and discuss the main challenges we face. Section III provides an overall picture of the presented approach, while in Section IV the solution is outlined. Specifically, it includes the formal framework and the implementation of the adaptation mechanism. Section V is devoted to the evaluation of the approach using the motivating example and the prototype implementation of the architecture. In Section VI we discuss related work and conclude the paper with open issues we plan to address in the future.
II. PROBLEM STATEMENT

In this section we present the reference scenario from the logistics domain to illustrate the problem addressed in the paper. In this scenario a business process is implemented to handle the delivery of cars from ships to the retailers in the automobile terminal of the Bremen sea port [11]. Activities of the process include unloading the cars from the ship, storing, applying treatment procedures to meet the customer’s requirements and distributing to the retailers. This business process, depicted in Figure 1, is realized as an orchestration of appropriate services, which can be atomic (e.g., Car Check Service, Unloading Service, etc.) or composite (e.g., Store Car Service), they may exhibit non-deterministic behavior, and perform asynchronous message exchanges. Along with the services directly involved in the main process, additional various services are available in the application to deal with specific situations.

The business process described here is defined under an assumption that the involved activities are executed successfully leading to proper changes in the operated domain. However, numerous unexpected situations may take place at runtime. For example:

- A vehicle may be damaged while moving from the ship to the storage area. While the necessary activities for storage (e.g., ticket request) have already been activated, according to business policies defined for the domain the damaged cars cannot be stored, and the process fails.
- There is a queue at the treatment station. A vehicle has to be brought for treatment (according to specific customer requests) immediately after the registering at the storage area, but the treatment station is completely loaded and the procedure cannot be applied. While normally the availability of places is ensured by the storage service, some of the treatments could not be complete on time leading to the situation not foreseen by the process.

In the situations of these kind the process cannot proceed with normal activity execution and fails. In order to be able to recover business process at run-time, there is a need for appropriate process adaptation mechanisms. While there exist approaches that achieve adaptability by the use of predefined exception handlers, ECA-rules, or explicitly modelled process variants, in our settings they often fail. Indeed, in such a volatile context the set of possible situations to be considered at design time is too large and, moreover, unexpected situations still may take place.

Ultimately, what we need is a framework, which provides:

- **Context-awareness**: the role of the context is fundamental in realizing the adaptation activities [12] as it enables identifying when the process adaptation needed and what should be done. Specifically, we need 1) to model the context, 2) to relate services to context changes (to understand in which context they can be executed and how they affect the context), and 3) to observe the current process context.
- **Dynamic Adaptation**: to deal with numerous and often unforeseen changes, it is critical to be able to construct adaptation plan dynamically [13], without having all possible situations predefined in advance. In case of damaged car, based on the location of the process and the state of context, it should be possible to construct a new subprocess that involves evacuation service, car repairing, and finally bringing the vehicle to the storage area. In other words, we require dynamic that does not require any involvement of the process designers while executing and adapting the reference process nor needs hard-coded adaptation logics.

In this work we present an approach for dynamic process adaptation, based on automated service composition techniques. While the approach relies on the possibility to observe the business context and its changes, the specific monitoring techniques are out of scope of this paper.

III. OVERALL APPROACH

We present a novel adaptation approach that aims at addressing the problems outlined in Section II. The approach relies on the following key elements:

- **Context-aware model of the application.** We explicitly model the business context, in which the process operates, and describe how the services and policies of the application are related to the context and its changes.

- **Context-aware execution framework.** While executing business process, the process context is continuously monitored in order to check whether it changes as expected by the policies and service specifications.

- **Context-aware adaptation based on automated service composition.** If a critical context change is detected, the adaptation tries to recover the process. This is achieved through construction of a composite service that, starting from the actual context state, performs the necessary changes in the domain to bring the process and its context to the expected state.

1) **Context model:** At design time, along with the definition of a business process and its implementation on top of a set of available services, relevant contextual information is modeled. First, to model the context properties and their evolution, we define context property diagrams that capture possible values of the property (as the diagram states) and the changes of the property values (as transitions). Second, to model how the services change the context, we annotate them with the effects on the context properties that correspond to changes of the property value (i.e., to some of the transitions in the property diagram). In other words, the context may change due to the service execution or due to some spontaneous, “unexpected” events. Third, we also
capture the business policies over the services to state in which context setting the service may be executed. We do this annotating services with the preconditions on the context property values.

2) Execution framework: The reference architecture of our execution framework is depicted in Figure 2. The execution and adaptation of the reference process $M$ is controlled and coordinated by the Orchestrator component. The activities of the process are executed by the Process Engine component that exchanges messages with the external services. Every activity executed by Process Engine is synchronized with and controlled by Orchestrator; it receives the execution information $o$ (e.g., messages received, operations performed, etc.) and decides whether to proceed with the next execution step $i$ or to perform process adaptation.

The decision on adaptation is based on the contextual information provided by the Context Manager component.

Context Manager continuously monitors and updates the values of the context properties associated to the process instance$^1$. When the observed values of the context properties violates the preconditions of the next activity to execute, the process adaptation is initiated. Based on the current configuration of the context and of the process, Orchestrator derives the specific adaptation problem $\xi$ and sends the corresponding request to the Adaptor component. In response, Adaptor generates the new subprocess $M_{\text{adapt}}$, which aims to do its best to “recover”, so that the blocked activity can be executed and the main process can continue. Orchestrator starts the execution of the generated subprocess and then continues the execution of the main process. If this execution fails again due to exogenous contextual changes, a new round of adaptation is undertaken.

It is important to notice that our adaptation is completely run-time and automated. Furthermore, it does not require hard-coded adaptation logics: the specific solution is generated dynamically, based only on the current state and the available services. Another important aspect is that the approach constantly observes the context and can immediately react to the critical changes. In particular, even if during the execution of the adaptation process some other problem is detected, the process is immediately terminated and the new adaptation is requested.

3) Adaptation: The adaptation problem $\xi$ sent to Adaptor comprises the current status of the system (values of the context properties, state of the involved services), set of available services that may be used for adaptation, and the adaptation requirements. As we already mentioned, the primary objective of the adaptation is to “unblock” the

$^1$We remark that the specific techniques for the context monitoring is out of the scope of this paper; approaches like [14] may be exploited for this purpose.
process, i.e., to change the context such that the violated precondition of the executed activity is satisfied. To accomplish this, Adaptor generates a composition of those services $M_{adapt}$, which, being executed together, achieve the necessary effect on the context. The construction of the composition is performed with the use of automated planning techniques [15]: the service specifications, the model of context (i.e., context diagrams), and the goal specifications are transformed into the planning problem and the resulting plan is then transformed into the composed service.

It is important to note that for the adaptation we do not consider the whole model of the context and its evolution (which may also be incomplete as some unexpected changes may be unforeseen). Instead, we make an assumption that during the adaptation no exogenous context changes take place, but only those that are entailed by execution of the services involved in the adaptation process. This assumption brings the following advantages. First, we do not necessary need to define at design time all possible evolutions of context: only effects of the services are important. This allows us to react even to completely unexpected changes. Second, this drastically simplifies the planning problem and the construction of the adapting composition becomes very efficient. On the other hand, this assumption does not lead to incorrect results. Indeed, if while the execution of the generated composed process some exogenous contextual change takes place, the Orchestrator component will immediately react to this, triggering another adaptation. Given the fact that those exogenous changes aim at representing extraordinary situations and events, such adaptations should also happen rarely, and the recovery activities eventually terminate.

We also remark that as the services may be non-deterministic (e.g., the diagnosis and repair service may complete successfully or may fail if the care is heavily damaged), the primary objective is not guaranteed (e.g., there is no way to proceed with unrepairable vehicle). For such cases, it is possible to associate with the process some "finalizing" recovery goals (i.e., dispose the car) that should become secondary objectives of the adaptation process in such extreme situations.

IV. Solution

In this section we define formally the elements of the presented approach and the solution. In particular, we give the formalization of the business process context and its changes, of the constituent services and their relation to the context, and of the adaptation problem. We then present the solution to the adaptation problem in terms of service composition via planning.

A. The Formal Framework

1) Context Property: In our approach, we explicitly formalize the business context of the reference process as a set of context properties. A context property represents some important characteristic of the environment that can change over time. For example, in the logistics scenario context properties may be the location of the car, the status of the car (operable/damaged), the status of the treatment area (busy/occupied), etc. We model the evolution of a context property with a context property diagram, which is a state transition system. Here states correspond to possible configurations of a property and transitions stand for possible property evolutions. Each transition is labeled with an event that characterize the change.

**Definition IV.1 (Context Property Diagram)** Context property diagram $c$ is a tuple $\langle L, L_0, E, T \rangle$, where:
- $L$ is a set of configurations and $L_0 \subseteq L$ is a set of initial configurations;
- $E$ is a set of property-specific events;
- $T \subseteq L \times E \times L$ is a transition relation;

We denote with $L(c)$ and $E(c)$ the corresponding elements of context property diagram $c$.

It is important to note that the context property may evolve as an effect of service invocations (e.g., vehicle get repaired after the repair service is engaged), which corresponds to the “normal” behavior of the domain, but also as a result of volatile – “unexpected” – changes. In these regards we can distinguish controllable events, i.e., those that may be achieved through services, and uncontrollable events, i.e., exogenous events of the environment. With uncontrollable events we capture unexpected situations that may require process adaptation.

Normally, the context is rather complex and consists variety of context properties $C$. The state of the context is a product of states of its property diagrams.

**Example.** Context property diagram for the car status may be presented as follows:

The initial state of the diagram is $OK$ (the car is operable). Due to exogenous events, the car can get slightly or severely damaged. The corresponding transitions labeled with events $light\_damaged$ and $severe\_damage$ are presented. A damaged car can be repaired with the repairing service (transitions labeled with event $repaired$). Additionally, a severely damaged car can be recognized as unrepairable with that service (event $unrepairable$).
2) Service: In order to model complex service protocols that feature asynchronous, stateful and non-deterministic behavior (e.g., those specified in BPEL), we use state transition systems, where transitions correspond to service operations (input and output messages). To capture the impact of service execution on the domain and to represent the business policies, under which we expect services to operate, we annotate service transitions with context preconditions and effects. A precondition $P$ is a set of configurations of $C$, in which the execution of the service operation is allowed. We will use these preconditions to detect the need for adaptation. An effect $E$ is a set of property-specific events specified in $C$, which fire (and thus make $C$ evolve) when a corresponding service transition takes place. As such, effects show how the execution of services affects the context.

Definition IV.2 (Annotated Service) Annotated service $s$ is a tuple $(L, L_0, I, O, T)$, where:

- $L$ is a set of states and $L_0 \subseteq L$ is a set of initial states;
- $I$ is a set of input actions (receiving a message);
- $O$ is a set of output actions (sending a message);
- $T \subseteq L \times P^* \times A \times E^* \times L$ is a transition relation, where $A = I \cup O$ is a set of actions; $P = \prod_{c_i \in C} L(c_i)$ is a set of configurations of $C$, thus $P^*$ stands for a precondition; $E = \bigcup E(c_i), c_i \in C$ is a set of controllable events in $C$, thus $E^*$ stands for an effect.

We denote with $L(s)$, $I(s)$ and $O(s)$, the corresponding elements of annotated service $s$.

Example. The formalization of annotated Car Repair service is represented as follows:

![Diagram]

Here, a request (input action ?CarRepairRequest) is followed by a non-deterministic response (output actions !CarRepairOk and !CarRepairFailed). The preconditions define that the car should be at the treatment area when it is being repaired (precondition car_location $\in \{\text{TREATMENT}\}$) and that the service should be applied only to a broken car (precondition car_status $\in \{\text{DMG\_LIGHT, DMG\_SEVERE}\}$). The effects of the output transitions define that the car may be successfully repaired (event repaired, which brings the status diagram to the state OK) or is unrepairable (event unrepairable, bringing the property to the state DMG\_DISPOSED).

3) Adaptation Problem: The adaptation strategy we adopt in this paper is to recover from the process failure so that the process execution can be resumed from the point where it has been blocked.

Let us consider process $M$ that orchestrates a set of services $S$, whose preconditions defined over context property diagrams $C$. When the preconditions of the next activity (e.g., “move a car”) to execute are violated (e.g., the precondition is that the car is operable, but the car is damaged), the execution should not proceed and the adaptation of $M$ is required. In order to adapt reference process $M$ we generate adaptation process $M_{adapt}$, which is the orchestration of services $S$ that implements the above adaptation strategy. The primary goal of resuming the execution of $M$ can be expressed as the reachability of the configurations of services $S$ and context property diagrams $C$ in which the execution of the next activity is possible. As we already mentioned, it is also possible to specify additional configurations to which the process should be brought when the above adaptation cannot be achieved. We remark that those recovery configurations are terminal for the process. We use them as a secondary (i.e., less preferable) goal for our adaptation problem. For example, we may require the car to be disposed, when its repair is not possible anymore.

Formally, adaptation problem may be defined as follows.

Definition IV.3 (Adaptation Problem) Adaptation problem $\xi$ is a tuple $(C, S, I, G_I, G_{II})$, where:

- $C$ is a set of context property diagrams;
- $S$ is a set of services defined over $C$;
- $I_i \subseteq L(s_1) \times \ldots \times L(s_m) \times L(c_1) \times \ldots \times L(c_n), s_i \in S, c_j \in C$ is the current configuration of context property diagrams $C$ and annotated services $S$;
- $G_I, G_{II} \subseteq L(s_1) \times \ldots \times L(s_m) \times L(c_1) \times \ldots \times L(c_n), s_i \in S, c_j \in C$ are sets of primary and secondary goal configurations;

We denote with $C(\xi)$, $S(\xi)$, $I(\xi)$, $G_I(\xi)$, $G_{II}(\xi)$ the corresponding elements of the adaptation problem $\xi$.

The solution to adaptation problem $\xi$ is process $M_{adapt}$ that orchestrates services $S(\xi)$. When executed from current configuration $I(\xi)$, $M_{adapt}$ brings services $S(\xi)$ and context property diagrams $C$ to one of primary goal configurations $G_I(\xi)$, otherwise to one of secondary goal configurations $G_{II}(\xi)$. We remark that $M_{adapt}$ succeeds only if no uncontrollable events happen during its execution.

B. Adaptation Strategy and Derivation of an Adaptation Process

In order to automatically solve adaptation problems, we use the variation of the service composition approach presented in [15]. According to it, a composition problem
is transformed into a planning problem and the planning techniques are used to resolve it. So do we.

The planning domain is derived from the adaptive problem. In particular, a set of \( n \) services \( (s_1, \ldots, s_n) \) and \( m \) context property diagrams \( (c_1, \ldots, c_m) \) are transformed into STSs using pretty straightforward transformation rules very similar to those presented in [15]. Before this transformation, all transitions labelled with uncontrollable events are removed from context property diagrams (so we plan in "controllable" environment).

So we get STSs \( \Sigma s_1, \ldots, \Sigma s_n \) and \( \Sigma c_1, \ldots, \Sigma c_m \). The planning domain \( \Sigma \) is a product of all STSs of the annotated services and context property diagrams synchronized on preconditions and effects:

\[
\Sigma = \Sigma s_1 \parallel \ldots \parallel \Sigma s_n \parallel \Sigma c_1 \parallel \ldots \parallel \Sigma c_m
\]

Initial state \( r \) of the planning domain is derived from the current configuration \( I(\xi) \) of the adaptive system, in which the need for the adaptation aroused.

Finally, the sets of primary and secondary goal configurations \( (G_I \) and \( G_{II} \) respectively) are transformed into the sets of configurations \( G_I^\Sigma \) and \( G_{II}^\Sigma \) of the planning domain \( \Sigma \). We denote the planning goal as a reachability goal with preferences:

\[
\rho = (G_I^\Sigma, G_{II}^\Sigma)
\]

which means that the primary planning goal is to reach one of the states in \( G_I^\Sigma \), and the secondary goal is to reach one of the states in \( G_{II}^\Sigma \).

After all, we apply the approach of [16] to domain \( \Sigma \) and planning goal \( \rho \) and generate a controller \( \Sigma_c \) (plan), which is such that \( \Sigma_c \triangleright \Sigma \models \rho \) (domain \( \Sigma \) reaches goal \( \rho \) when controlled by \( \Sigma_c \)). The state transition system \( \Sigma_c \) is further translated into executable process \( M_{adapt} \), which implements the above described adaptation strategy. The back translation from STS into executable specification is quite simple: input actions in \( \Sigma_c \) model an incoming message from a component service, while output actions in \( \Sigma_c \) model an outgoing message to a component service.

**Correctness of the approach.** The proof of the correctness of the approach consists in showing that, under the aforementioned assumptions, all the executions of the adaptation process \( M_{adapt} \) (translation of controller \( \Sigma_c \)) implement the adaptation strategy. Here we outline the key points of the proof. It is easy to see that each execution \( \theta \) of the adaptation process is also a run of the domain, i.e., if \( \theta \in \Pi(\Sigma_c) \) then \( \theta \in \Pi(\Sigma) \). Under the requirement that all the executions of the domain terminate in goal states, we get that the executions of the domain implements the adaptation strategy. As a consequence, the following theorem holds.

**Theorem IV.1 (Correctness of the approach)** Let:

- \( \Sigma s_1, \ldots, \Sigma s_n \) be the STS encoding of services \( s_1, \ldots, s_n \) and
- \( \Sigma c_1, \ldots, \Sigma c_m \) be the STS encoding of context property diagrams \( c_1, \ldots, c_m \).

Let \( \Sigma_c \) be the controller for a particular composition problem

\[
\Sigma = \Sigma s_1 \parallel \ldots \parallel \Sigma s_n \parallel \Sigma c_1 \parallel \ldots \parallel \Sigma c_m
\]

\[
\rho = (G_I^\Sigma, G_{II}^\Sigma)
\]

i.e., \( \Sigma_c \triangleright \Sigma \models \rho \). Then the executions \( \Pi(\Sigma_c) \) implements the adaptation strategy.

As we mentioned in Section III, we assume that no exogenous context changes may take place during the adaptation, and therefore in the transformation of the adaptive system into a planning domain we ignore all uncontrollable transitions in context property diagrams. Our experiments show that such an approach leads to very efficient adaptation, as it significantly increases the performance of the planning algorithm by reducing the amount of reachable states in the planning domain. Furthermore, in many domains external contextual changes are quite improbable so that additional rounds of adaptation are rarely needed. Compared to [16], we also use preprocessing of the planning domain by pruning unreachable states, which also increases the performance of the planning algorithm.

**V. Evaluation**

In order to evaluate our approach we implemented the prototype of the architecture presented in Section III and designed the reference scenario using the formal framework introduced in Section IV-A.

In order to formalize the reference scenario with enough details, we distinguished and specified 10 context property diagrams (car location, car status, treatment areas status etc.) and 16 stateful services (services for moving, storing, repairing, treating, checking the car etc.). Service specifications were annotated with preconditions and effects over the context property diagrams. In the initial state the car is on the ship and it is operable. The goal state is where the operable car is properly treated and is loaded on the ship. After the delivery, the alternative goal state is where the car is unrepairable and at the disposal area. We also specified external context changes that model unexpected light and severe damage of the car and switching of the treatment areas status from “occupied” to “free” and vice versa. Finally an implementation of the reference process was specified.

The implementation of the reference process is quite complex and dozens of different unexpected situations, each requiring specific adaptation, may happen. We will consider only those two mentioned in Section II.
A. Damage on the move

In this example, the car is severely damaged while moving towards the storage area. There is a storage ticket reserved for the car, which guarantees a place at the storage area. This situation requires adaptation since it is not considered in the reference process. There is a number of business policies that are implemented in the framework using action preconditions:

- The car can move on its own only if it is not severely damaged;
- The car can be stored only if it is not damaged;
- The car can be repaired only if it has a storage ticket;
- The car can be repaired only at the treatment area;
- The car must have no storage ticket when it is being repaired (not to waste the space at the storage area);
- The storage ticket can be received only at the terminal;

The goal of the adaptation is to reach the configuration “the operable car is moving towards the storage with the storage ticket in hands”. The recovery goal is to leave the car at the treatment area if it is unrepairable. The adaptation process \( M_{\text{adapt}} \) obtained (Fig. 3) is completely compliant with the context, business policies and goals (the primary goal is depicted as success and the secondary goal as failure). We remark that the same severe damage in different context settings could result in absolutely different adaptation process generated.

![Figure 3: Adaptation process for “Damage on the Move”](image)

B. Waiting for treatment

The car has just moved to the treatment area and is ready to be treated. The treatment consists of three operations: equipment, painting and cleaning. It turns out that the equipment and painting areas are busy. This is an unexpected situation and the adaptation is undertaken. The following policies play role for this case:

- The car should complete all three treatment operations before leaving the treatment;
- Cleaning has to be performed after painting;

The adaptation process (Fig. 4) suggests to use a special notification service that notifies the client about the treatment areas just freed. One can see, that the adaptation is successful only if we receive a message about painting or equipment areas to be free. The cleaning area is not of interest for the moment, since cleaning cannot be performed before painting.

We ran the examples on the double-core 2Ghz Windows XP laptop with 4 Gb of RAM. The derivation of the two adaptation processes took 13 and 3 seconds respectively.

While analyzing these numbers we have to realize that although the adaptation processes presented do not feature high complexity, they are derived from quite a large domain that requires significant resolution time. Taking into account the real complexity of the reference scenario, we consider these results as a proof of high potential of our approach, applicable in a vast majority of real application domains.

![Figure 4: Adaptation process for “Waiting for Treatment”](image)

VI. RELATED WORK AND CONCLUSION

Various frameworks can be found in the literature with the objective to support adaptation of business processes, each of them addressing a specific issue. When the set of possible adaptation configurations is fixed and known a priori, it is possible to completely specify them at design time. In this case, we talk about \textit{built-in adaptation} and the adaptation may be performed using one of the following approaches: by extending standard notations as BPEL with the adaptation-specific tools [3], [4] using a set of predefined adaptation rules [5], [6], [17], using aspect-oriented approaches [8], [9], [18], or modeling and managing business process variants [10], [18], [19]. All the aforementioned approaches are able to capture a precise set of exceptional events or situations and to use for each of them a predefined adaptation rule. In this paper, we have presented a novel approach to adapt business processes where the running application and the adaptation logic are two separate components and the set of possible adaptations are generated directly at runtime when a problem arises (i.e., \textit{dynamic adaptation} [13]). This means that it is not necessary to know at design time which actual conditions will trigger the adaptation, and which kind of adaptation should be performed to correct it. To understand when, how, and why a business process should be adapted we have modeled the execution environment in such a way that its changes are synchronized with the application execution and vice versa. When we need to execute adaptation, this two-way synchronization let the adaptation tool understand what the current state of the business process and the current context are. At the same time, using the current context and its evolution model, we are able to provide the adaptation engine with the precise goal to reach. The adaptation engine generates the adaptation process to execute and brings the main process back to a state from where its execution can be successfully resumed.

A framework similar to our has been proposed in [20]. It is called SmartPM and is able to adapt processes in...
case of unforeseen events. However, we consider our formal framework to be much more intuitive and easy to use for the designer. Moreover, the approach in [20] can work with abstract services and does not allow for using real services specified in one of standard languages. On the contrary, our approach is developed for stateful and non-deterministic services that can be modeled using such a language as BPEL and transformed into our formalism automatically.

In the future, we plan to extend our framework in order to allow for more sophisticated adaptation strategies that can be chosen automatically. At the same time, our adaptation mechanism has been defined to be applied to single instances of a process model. We want to use a set of adapted instances together with their execution history as training cases for evolution mechanisms in order to progressively improve process model that may be further used if a new process instance is to be instantiated.

Acknowledgements.

The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (SC-Cube) and from the FP7 EU-FET project 213339 ALLOW.

References

Exploiting Codified User Task Knowledge to Discover Services

K. Zachos, A. Kounkou, and N.A.M. Maiden, City University

Abstract—Most methods and techniques for engineering service-based applications do not explicitly exploit knowledge about users and their tasks. However, codified knowledge about user tasks can be applied to improve service discovery and composition so that it is adapted to these tasks. This paper reports the application of user task models to improve requirements-based service discovery. It describes how user task models can be applied to enhance service discovery based on a catalogue of user tasks. More specifically it reports an evaluation of a new tool that demonstrate its potential utility when improving the discovery of web services for an e-government service-based application.

Index Terms—Web Services Discovery, Services Discovery Engine, user task models.

1 INTRODUCTION

SERVICE-oriented computing has experienced a marked uptake across a variety of application domains over the past decade. As a result, it is becoming increasingly important for its continuing success to address not only technological, but also human and organizational aspects [1]. Current service-oriented approaches primarily rely on business process models and notations such as BPEL to indicate the process-oriented context in which a service needs to be invoked. However, process models normally lack important information about the actor performing the process and their goals, actions and constraints. Although initiatives such as BPEL4People [3] attempt to incorporate human considerations into the specification of business processes, they are limited to describing human activities as simple processes without addressing knowledge about users and their tasks. Processes and techniques for service discovery have been the subject of extensive research in the past [2]; however, the application of user task knowledge to refine the discovery and selection of best-fit services is yet to be demonstrated.

In this paper we report the development, application and evaluation of user task models for service discovery. A collection of domain-independent user task models was developed to populate an on-line catalogue accessible by one service discovery engine - the EDDiE engine [4] previously developed by the EU-funded SeCSE consortium. The user task models encapsulate important knowledge normally not available to service discovery engines about user goals, tasks and resources, which EDDiE exploits to match corresponding user tasks and discover fitting services. The application of user task models to reformulate service queries has, we conjecture, the potential to improve the levels of precision and recall of discovered services. Removing user task terms less that are likely to match to service description terms from service queries can increase precision, whilst adding terms about service classes more likely to match to service description terms can increase recall. Therefore, we evaluated the effect of service query reformulation with user task models to investigate 4 hypotheses:

H1: Extending rather than replacing service queries with additional knowledge about user tasks will improve the overall effectiveness of services.

H2: The reformulation of service queries with knowledge about user tasks will decrease the number of irrelevant services retrieved by the EDDiE service discovery engine;

H3: The reformulation of service queries with knowledge about user tasks will increase the number of relevant services retrieved by the EDDiE service discovery engine;

H4: The reformulation of service queries with knowledge about user tasks will improve the overall correctness of services retrieved by the EDDiE service discovery engine.

Hypothesis H1 relates to the different query reformulation strategies of the new task-based service discovery approach used to incorporate user task models in the service discovery process. In particular, each service query can either be extended or replaced with new text describing user task knowledge. Therefore, we firstly evaluated the effect of each query reformulation strategy within the task-based service discovery approach.

Results from the evaluation studies revealed that extending rather than replacing service queries with user task knowledge improves the overall effectiveness of discovered services. Results also indicated mixed support for H2, H3 and H4. Data produced and analyzed with service queries that were generated with EDDiE’s query expansion capability resulted in the acceptance of only hypotheses H2 and H3, whereas findings led to an acceptance of all three hypotheses for non-expanded service queries.

The remainder of this paper is in 5 sections. Section 2 describes existing user task models and their use in SBA engineering. Section 3 describes the task-based
service discovery solution and demonstrates its key elements using a simple example. Section 4 reports results from a multiphase evaluation study that provided data with which to answer our hypothesis. The paper ends with discussions and future research directions.

2 User Task Models for SBA Engineering

In this paper we draw on previous research in human-computer interaction (HCI) and treat a user task model as a description of structured activities that are often executed during the interaction with a system, influenced by its contextual environment, and performed to attain goals [5, 6].

User task models have been used in various approaches to support different phases of the software development life cycle such as requirement analysis and usability evaluation. For instance, they help understanding how people perform their work [7] and so can function as a requirements elicitation device, and indicate how activities should be performed in an application being designed. They also allow for the software design to be described more formally, analyzed in terms of usability, and be better communicated to people other than the analysts [9]. Task models have semantically richer models and notations than processes currently used in BPEL4People; we conjecture they may hence provide more context-specific information for service discovery and composition. Indeed their specific semantics permit different possible uses of task models in service-centric systems; for example tasks and operations in HTA [10] can map to specific service capabilities, thus enabling finer-grain service discovery and selection. Likewise, the distinction between abstract, interaction, application and user tasks in CTT [11] can be mapped to different service types, thus informing more effective service composition, especially if finer-grain services are available.

Despite the potential importance of user task models for service-centric systems, research on the topic seems scarce and only few published studies could be found, none of which brought elements of support to our hypotheses. For example Paterno et al. [12, 13] describe an environment to support tasks and services matching on the basis of CTT task models. However, the association between tasks and services is manual and its application restricted to user interface development. Ruiz et al. [14] propose a methodological guide to the design of web services that uses user task descriptions analysis to identify web application’s required operations. This approach however, although automated, does not extend to the service discovery process or the quality of selected candidate services. More recently, Kritikos and Paterno [15] propose a model-driven approach using a domain ontology and a designer-provided task model to produce a service model specifying service combinations that realize the application’s functionality. As part of the process, service discovery queries are generated from each system task description and executed to discover, categorize and rank matching services based on their textual similarity measure. This approach however requires designers to produce models for each envisaged system; further, no information is provided on the impact of the task-based queries on service discovery. It is hence difficult to assess and contrast the performance of task-based service queries with that of standard queries in terms of number and relevance of discovered services for instance.

One goal of our research was to codify knowledge from HCI research about classes of user task. To do this we described domain tasks then manually generated reusable, domain-independent task models, similar to knowledge modeling in KADS [16]. For example a description of the task “go to somewhere” describes a general process of moving from a starting point to a destination, and knowledge about this task can be reused in different travel domains.

Overall, the role of user task knowledge in service discovery is to overcome the semantic mismatch problem that arises because the problem query and solution service are inevitably expressed using different ontologies – terms used to describe a user’s goals and tasks are often different to the terms used to describe the services that could be invoked usefully during these tasks. Therefore each user task model links knowledge about classes of service solutions to classes of known user tasks. The task-based service discovery algorithm can use these links between knowledge about the user’s problem domain and the solution space of classes of service to reformulate a service query – terms describing the user’s goals, tasks and resources, which are less likely to match to terms describing relevant services, are extended or replaced with terms that describe classes of relevant software services. For example, a task model describing the journey undertaken by a driver to reach a destination would replace terms in service queries such as plan, travel, route and help (terms in the domain of the user task) with terms such as journey planner, routing, parking space, and global positioning system (terms describing relevant classes of software service for the user task).

3 Codifying User Task Models during Query Reformulation

In this section we describe how we associated user task models, software services and service-based applications, and demonstrate key elements of service discovery using an example in which citizens utilize e-government services within the S-Cube Network of Excellence [16]. The actors involved in the example are an end-user who queries and uses a government journey planning service, and a public body that provides the service. The end-user queries journey details using the start and end points and travel preferences, then receives personalized route suggestions including relevant travel alerts and dynamic re-mapping of the route.

We opted to use the CTT task modeling formalism [6] to represent knowledge about user tasks because it adapts an engineering approach to user task models to overcome the limitations of other approaches with limited tool support or unsuitable notations with limited operators. We
selected the SeCSE service discovery environment for service-based systems upon which to design and implement the codified user task knowledge. The FP6 SeCSE Integrated Project is one of the cornerstone research development projects in service-centric systems funded by the European Commission. It has produced substantial research, development and evaluation results, as well as tool suites available to be extended in S-Cube. We extended SeCSE service queries specified using XQuery and selection filters with types of knowledge about user task knowledge.

3.1 Service Discovery in SeCSE

The EDDiE algorithm was extended to incorporate user task knowledge in the service discovery process. Previous research – SWWS, DIP, ASG, InfraWebs and NeP4B – demonstrated that semantic descriptions such as SAWSDL [18], MicroWSMO [19], WSMO [20] and OWL-S [21] can be used to enable more precise discovery of services. However, although these approaches focus on automation support and flexible matching between descriptions of services and queries, they do not address the problem of the heterogeneity of the vocabularies used in service descriptions and requirement specifications. To overcome the above-mentioned challenge, EDDiE uses established information retrieval techniques by formulating service queries from use case and requirements specifications expressed in structured natural language [22]. A full description is provided in [26].

The EDDiE algorithm has 4 key components; the Natural Language Processing, Word Sense Disambiguation, Query Expansion and the Matching Engine. In the first the service query is divided into sentences, then tokenized and part-of-speech tagged and modified to include each term’s morphological root (e.g. driving to drive, and drivers to driver). Secondly, the algorithm applies procedures to disambiguate each term by defining its correct sense and tagging it with that sense (e.g. defining a driver to be a vehicle rather than a type of golf club). Thirdly, the algorithm expands each term with other terms that have similar meaning according to the tagged sense, to increase the likelihood of a match with a service description (e.g. the term driver is synonymous with the term motorist which is also then included in the query). In the fourth component the algorithm matches all expanded and sense-tagged query terms to a similar set of terms that describe each candidate service, expressed using the service description facet [23], in the SeCSE service registry. Query matching is in 2 steps: (i) XQuery text-searching functions to discover an initial set of services descriptions that satisfy global search constraints; (ii) traditional vector-space model information retrieval, enhanced with WordNet, to further refine and assess the quality of the candidate service set. This two-step approach overcomes XQuery’s limited text-based search capabilities.

3.2 Task-Based Extension to Service Discovery

Query expansion alone cannot overcome the semantic mismatch between the problem query and solution service. To overcome the mismatch we extended the algorithm with a user task model catalogue that links knowledge about classes of service solution to classes of user tasks. The enhanced algorithm discovers services by matching terms describing the user problem to user task models that, in turn, are used to reformulate the service queries with terms describing classes of services in the task model. These reformulated service queries are then fired directly at service registries. We developed a prototype task model catalogue that we populated with domain-independent task models that covered the e-government domain. Examples of these user task models included health advise, calculate distance and request route.

3.2.1 Structure of a User Task Model

A user task model defines a reusable and generic task structure that encapsulates a well-defined functionality for a recurrent design problem in task modeling [10]. In the S-Cube Network of Excellence we use this definition to describe: (i) classes of tasks that re-occur during the use of service-based applications, and: (ii) classes of candidate service solutions demonstrated to solve these tasks. Fig.1 presents the schema of a user task model in the task model catalogue. Each user task model is an aggregation of information about the user task and classes of software service that can invoked to support the task. Each is described in turn.

Information about the user task includes a structured natural language description of the task in context and the task structure expressed in CTT [7]. Other attributes include a text natural language description of the user goal for the task. Each task can be one of four different types: (i) abstract tasks that are decomposed further; (ii) user tasks undertaken by the user; (iii) interaction tasks that are carried out by the interaction of a user with a software system; and: (iv) application tasks fully undertaken by software [11]. The sequence of different task is described by operators [7]. Each task can also contain one or
more resources that are needed for fulfilling the task. To be effective these descriptions of a task need to be specific to the user task but independent of different domains in which the task can be undertaken to be applied within and across these domains. Fig. 2 depicts part of the CTT task model of one of the user task models from our prototype task model catalogue that describes the need to calculate the distance to a location. The task model depicts both user sub-tasks such as enter destination and submit data, and application tasks that can be undertaken by invoked services such as check match with known locations and confirm data validity.

Classes of software service describe in structured natural language the functionality and operations of candidate classes of software that, based on past experience, support users to achieve their task if invoked. During query reformulation terms from the service class description matched to the user task model are used to constitute new queries fired at service registries. Table 1 reports one such class – DataValidation – that an application might invoke. Using a name (corresponding to a concept name), functional description (corresponding to the concept definition) and list of operations (derived from the action terms or verbs contained in the concept description).

The natural language data populating these classes was sourced from online encyclopedias selected for the breadth of terms they covered: primarily Wikipedia [30], then if a concept definition was not found, respectively: from the Encyclopedia Britannica [31] or finally, the more specialized Webopedia computer dictionary [32]. Each user tasks and subtasks were linked to at least one service class (as appropriate), with each service class possibly being mapped to many subtasks. For instance the subtasks “validate query” and “validate scale” were both mapped to the “data validation” and “data type checks” service classes.

The next section describes T-EDDiE – the new task-based service discovery approach, in more detail.

3.2.2 Task-Based Service Discovery process (T-EDDiE)

As Fig. 3 shows we extended the current SeCSE’s service discovery algorithm by adding a task model catalogue (Task KB) and two new components – the task navigator and query reformulator – to EDDiE.
queries that are expressed in terms of the service features rather than consumer requirements. The analyst can then modify these reformulated queries as needed;

3. **Service match:** the service discovery algorithm uses each revised service query to discover candidate service specifications in service registries. The result is an ordered set of service specifications that match to the revised service query.

T-EDDiE’s query reformulation algorithm uses one of the following 4 query reformulation strategies (QRS):

1. **Extend** service query with user task knowledge *without* applying any term expansion;
2. **Extend** service query with user task knowledge *with* the application of term expansion;
3. **Replace** service query with user task knowledge *without* applying any term expansion;
4. **Replace** service query with user task knowledge *with* the application of term expansion.

Fig. 4 shows all QRS for EDDiE and T-EDDiE that will be used in the evaluation studies.

Fig. 4. Tree view of EDDiE and T-EDDiE reformulation strategies

T-EDDiE’s query reformulation algorithm as well as the user task model selection process are described at length in [rejected ICSOC’10 paper]. The following section reports results from evaluations of T-EDDiE following on from recent successful evaluations and studies of SeCSE’s existing service discovery tools such as EDDiE [26] and AnTiQue [27]. We conducted these evaluations to determine whether or not they support the four hypotheses.

4 **T-EDDiE’s Summative Evaluation**

The purpose of this evaluation was to undertake a summative evaluation of the T-EDDiE algorithm and investigate our four hypotheses using precision, recall and balanced F-score statistical measures. The evaluation was carried out in four stages.

In the first stage, user task models were extended with knowledge about classes of software service. In the second stage we undertook a human assessment of candidate web services relevant to the government journey planning use case used to undertake the summative evaluation. In the third stage we compared the statistical measures of T-EDDiE’s QRS to determine which strategy is performing best when incorporating user task models in the service discovery process. In the final stage we compared the statistical measures of T-EDDiE against the original EDDiE algorithm.

For the latter two stages we used candidate web service populated in the SeCSE service registry [23] containing 215 existing web services in domains such as flight booking, weather reporting and route planning.

4.1 Extending User Task Models with Classes of Software Services

To evaluate the T-EDDiE algorithm it was essential to populate the task model catalogue with a small set of codified user task models. User task models were specified for the following six tasks: obtain health advice; access a patient record; request a route; calculate a distance; park a car; zoom into a map. For each user task we described the task in natural language and specified one CTT task model. Next, to use these task models to reformulate service queries, we developed classes describing candidate service solutions – i.e. software service concepts proven to execute these tasks – to be associated with the user task models. In total, 33 service classes were described and linked to the user task models as described in Section 3.2.1.

4.2 Classification of Candidate Services

We used human judgment to determine whether or not candidate web services from the SeCSE service registry were relevant to the government journey planning use case – partly shown in Table 2 – which was used to undertake the summative evaluation. We conducted a controlled study with 6 human judges – experts in knowledge of software services applied to e-government – who assessed the relevancy of each web service description with regards to the journey planning use case. The experts found 37 web services as being relevant to our use case, and their judgment provided the baseline with which to assess both EDDiE and T-EDDiE service discovery algorithms.

2 The balanced F-score assesses the effectiveness of both query mechanisms by identifying the harmonic mean of the precision and recall where recall and precision are evenly weighted.
4.3 Evaluating T-EDDiE’s Query Refomulation Strategies

Before investigating the effectiveness of task-based service discovery, we evaluated which QRS is better suited when attempting to incorporate user task models in the service discovery process. We therefore investigated T-EDDiE’s performance based on the retrieval of web services judged as relevant with regards to the journey planning use case. To do this we generated and fired 44 separate queries – 11 queries for each QRS as described in Section 3.2.2. – containing parts of the journey planning use case at the service registry. For all QRS we then computed scores for precision, recall and balanced F-score. Precision was defined as:

\[
\text{Precision} = \left( \frac{\text{Total retrieved relevant services}}{\text{Total discovered services}} \right) \times 100
\]

Recall was defined as:

\[
\text{Recall} = \left( \frac{\text{Total retrieved relevant services}}{\text{Total classified relevant services}} \right) \times 100
\]

The balanced F-score was defined as:

\[
\text{F-score} = \left( \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \right) \times 100
\]

Results for each QRS are shown in Table 3.

The arithmetic mean was used to calculate the average of each query set. No significant differences were obtained for the precision scores, i.e. the precision scores for both extend QRS were 44% and 38% respectively, whereas the precision scores for both replace QRS were 42% and 39% respectively. With regards to the recall, both scores for the extend QRS were much higher than those for the replace QRS, 58% and 65% compared to 27% and 43% respectively. A similar pattern revealed the F-scores – 49% and 47% compared to 29% and 36% respectively.

4.3.1 Evidence to Support Research Hypothesis H1

We used results from the above evaluation to determine whether the findings led to an acceptance of hypothesis H1 (Extending rather than replacing service queries will improve the overall effectiveness of services). Although results indicate better performance when extending rather than replacing service queries with user task knowledge, to determine whether the measurements of the examined sample are representative of the population – i.e. the results are statistically valid and the probability of obtaining the observed difference is low – we performed paired t-tests on both extend and replace QRS, with and without the application of term expansion. We performed paired t-tests on each QRS pair in order to compute p-values to either reject or accept the null hypothesis, i.e. on average there is no difference between extending or replacing service queries user task knowledge. Results are shown in Table 4 and Table 5.

<table>
<thead>
<tr>
<th>TABLE 2</th>
<th>THE JOURNEY PLANNING USE CASE SPECIFICATION USED IN THE EVALUATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Case</td>
<td>Journey Planning</td>
</tr>
<tr>
<td>Privy</td>
<td>A user requests the e-Government route planning service. The user sends a journey planning request with details about the start, end point and travel preferences for his journey. The user confirms his current location as the starting point. The user inputs his destination address and his travel preferences. The system displays personalized suggestions of routes to follow which the user can query for additional details. The user chooses a route and follows the route towards the underground station. The user verifies his direction on the area map on his device. The user arrives at the underground station, gets on the train and finally arrives at his destination.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 3</th>
<th>T-EDDIE’S QRS OVERALL PERFORMANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall Performance (k-mean)</td>
<td>Extend</td>
</tr>
<tr>
<td>Total</td>
<td>50</td>
</tr>
<tr>
<td>Relevant</td>
<td>21</td>
</tr>
<tr>
<td>Non-Relevant</td>
<td>29</td>
</tr>
<tr>
<td>Precision</td>
<td>44</td>
</tr>
<tr>
<td>Recall</td>
<td>58</td>
</tr>
<tr>
<td>F-score</td>
<td>49</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 4</th>
<th>PAIRED T-TESTS ON BOTH EXTEND AND REPLACE QRS USING NO TERM EXPANSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>QRS</td>
<td>P-value</td>
</tr>
<tr>
<td>Total</td>
<td>0.0004</td>
</tr>
<tr>
<td>Relevant</td>
<td>0.0001</td>
</tr>
<tr>
<td>Non-Relevant</td>
<td>0.0096</td>
</tr>
<tr>
<td>Precision</td>
<td>0.5513</td>
</tr>
<tr>
<td>Recall</td>
<td>0.0001</td>
</tr>
<tr>
<td>F-score</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE 5</th>
<th>PAIRED T-TESTS ON BOTH EXTEND AND REPLACE QRS USING TERM EXPANSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>QRS with expansion</td>
<td>P-value</td>
</tr>
<tr>
<td>Total</td>
<td>0.0012</td>
</tr>
<tr>
<td>Relevant</td>
<td>0.0025</td>
</tr>
<tr>
<td>Non-Relevant</td>
<td>0.0027</td>
</tr>
<tr>
<td>Precision</td>
<td>0.5726</td>
</tr>
<tr>
<td>Recall</td>
<td>0.0023</td>
</tr>
<tr>
<td>F-score</td>
<td>0.0229</td>
</tr>
</tbody>
</table>
Results from these paired t-tests indicate that there is no significant differences with regards to precision, whereas there is enough evidence to suggest a difference, on average, between extend and replace QRS on all other dimension, including recall and F-score. Although on the surface, the precision was an inconclusive result; H1 was accepted because of both recall and the balanced F-score measure that revealed a superior value for the extend QRS.

### 4.4 EDDIE vs. T-EDDIE

Having identified that extending rather than replacing service queries is better suited when attempting to incorporate user task models in the service discovery process, we investigated the performance of both EDDIE and T-EDDIE based on the retrieval of web services judged as relevant with regards to the same journey planning use case. To do this we fired two original EDDIE service queries (with and without term expansion) and 22 task-based service queries extended with user task knowledge (with and without term expansion) – containing parts of the journey planning use case at the service registry. It is worth noting that the number of generated EDDIE service queries was much lower compared to the number of generated T-EDDIE service queries – this is due to the fact that T-EDDIE has a greater number of QRS compared to EDDIE (see Fig. 4), and because each user task model consist of multiple sub-tasks that are used to generate different service queries each time.

Totals of relevant and non-relevant web services retrieved by EDDIE and T-EDDIE are shown in the first three rows of Table 6. For both approaches we again computed scores for (1) precision to investigate whether T-EDDIE decreases the number of irrelevant services; (2) recall to investigate whether T-EDDIE increases the number of relevant services; and (3) balanced F-score to investigate whether T-EDDIE improves the overall correctness of services. Table 6 shows a summary of the results where the calculated means are used for each set of T-EDDIE queries.

<table>
<thead>
<tr>
<th>Overall Performance (Mean)</th>
<th>No Expansion</th>
<th>Expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EDDIE</td>
<td>T-EDDIE</td>
</tr>
<tr>
<td>Total</td>
<td>18</td>
<td>50</td>
</tr>
<tr>
<td>Relevant</td>
<td>11</td>
<td>21</td>
</tr>
<tr>
<td>Non-Relevant</td>
<td>7</td>
<td>29</td>
</tr>
<tr>
<td>Precision</td>
<td>61</td>
<td>44</td>
</tr>
<tr>
<td>Recall</td>
<td>30</td>
<td>58</td>
</tr>
<tr>
<td>F-score</td>
<td>40</td>
<td>49</td>
</tr>
</tbody>
</table>

As far as queries without term expansion were concerned, the precision score for EDDIE and T-EDDIE was 61% and 44% respectively. EDDIE and T-EDDIE retrieved 11 and 21 relevant services respectively, so the recall score was 30% and 58% respectively. The F-score for EDDIE and T-EDDIE was 40% and 49% respectively. Similar results were observed for service queries that used term expansion with the exception of the F-score measure. The precision score for EDDIE and T-EDDIE was 63% and 38% respectively. EDDIE and T-EDDIE retrieved 17 and 24 relevant services respectively, so the recall score was 46% and 65% respectively. However, the F-score for EDDIE and T-EDDIE was 53% and 47% respectively.

As both service discovery engines return a ranked sequence of services, we investigated the order in which the discovered – both relevant and non-relevant – services were presented. We computed another performance measure – mean average precision – that emphasizes ranking relevant services higher. The resulting values for both EDDIE queries were 63% and 67% respectively. The values for term expanded and non-expanded T-EDDIE queries sets were 50% and 48% respectively. A deeper analysis of the top 10 discovered services for each service-set supports the mean average precision values as T-EDDIE and EDDIE discovered on average 5 and 3 non-relevant services respectively.

#### 4.4.1 Evidence to Support Research Hypotheses H2, H3 and H4

We used results from the summative evaluation to determine whether the findings led to an acceptance of the hypotheses H2-H4:

- **H2 (T-EDDIE will decrease the number of irrelevant services)** was rejected as both precision and mean average precision of T-EDDIE’s results were lower than the equivalent scores for EDDIE.
- **H3 (T-EDDIE will increase the number of relevant services)** was accepted due to the fact that T-EDDIE’s recall score was indeed significantly higher than EDDIE’s recall value.
- **H4 (T-EDDIE will improve the overall correctness of services)** was accepted but only for non-expanded queries because of the balanced F-score measure that revealed a superior value for T-EDDIE.

Although on the surface, the precision was a negative result, T-EDDIE was able to retrieve on average 58% and 65% of services respectively that the experts judged as relevant – 28% and 19% better performance than EDDIE. The so-called inverse relation between precision and recall may partly explain the low precision score; indeed, previous studies have reported on a tendency for precision to decrease as recall increases (e.g. [28] and more recently, [29]), which is the trend observed in our study. For the specific purpose of service discovery, we believe the relative importance of precision over recall (and vice-versa) is subject to the scenarios of use under which the discovery process is performed.

Of course there are numerous threats to the validity of the reported results. One threat to the results validity was identified and comprised the randomness and size of the tasks and service samples selected for this study. It is also acknowledged that the study participants were constrained to exclusively rely on their expertise and the textual descriptions provided, whereas service testing and code examination are often part of the process of service evaluation for set purposes. Another threat to the conclusion validity of the evaluation results is the sample size for the second part of
the evaluation – 1 use case specification fired at 1 registry. However the current small body of research into codified HCI knowledge for SBAs and the absence of any research into user task models to enhance the service discovery process led us to run a formative-predictive evaluation to generate a first set of results to explore T-EDDiE’s feasibility then provide a framework and focus for more subsequent rigorous evaluation. Finally, one threat to the external validity of the results might have been the choice of domain. The results have external validity if we can generalize them outside of government journey planning to other domains.

5 DISCUSSION AND FUTURE WORK

This paper explored the effect of codified user task knowledge on service discovery. Results from this first empirical investigation indicate future research directions in the S-Cube Network of Excellence. One such direction is to refine the T-EDDiE algorithm so that its computed match values better correspond to human judgments of similarity.

In S-Cube we conjecture that codified HCI design knowledge can be used to inform service composition during the architecture design for a SBA. Most existing business process and workflow modelling techniques model coarse-grain processes with little support for finer-grain user tasks of different types and interactions with the service-based applications. User task models from HCI naturally plug this gap, and introduce new concepts such as task goals from the user perspective not modelled using approaches such as BPEL. We plan to explore the proposal through extension of another SeCSE development tool – the Composition Designer. In future work we will extend the Composition Designer to allow a service integrator to generate a service composition with user task models in order to inform more effective service composition.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

REFERENCES

Konstantinos Zachos received a Computer Science Diploma degree from RWTH Aachen in 2004, and a PhD degree in Software Engineering (specialisation in Service-Oriented Requirements Engineering and Creativity) from City University London in 2008. He is a researcher at the Centre for Human Computer Interaction Design at City University and has been involved in research in the areas of requirements engineering, service discovery mechanism and techniques, and creative problem-solving and learning.

Angela Kounkou received a Computer Science Diploma degree from City University London in 2007, and a Masters degree in Human-Computer Interaction with Ergonomics from University College London in 2009. She is a researcher at the Centre for Human-Computer Interaction design at City University London and has been involved in research in the areas of developers’ information seeking behaviour and service discovery mechanisms and techniques.

Neil A. M. Maiden is a Professor of Systems Engineering, Head of the Centre for Human-Computer Interaction Design, and academic lead of the Centre for Creativity in Professional Practice at City University London. His research interests include establishing the requirements for complex socio-technical systems, scenario-based design, service-oriented systems and cloud computing, goal modelling, and technology-enhanced creative design. He also investigates new ways of delivering results from these research areas into practice. Neil has published over 150 peer-reviewed papers in academic journals, conferences and workshops proceedings. He was Program Chair for the 12th IEEE International Conference on Requirements Engineering in Kyoto in 2004. He is on the Editorial Boards of the IEEE Software and Requirements Engineering Journal and a former member of the Editorial Board for IEEE Transactions of Software Engineering. He is also the Editor of the IEEE Software’s Requirements column.
Towards Proactive Adaptation: A Journey along the S-Cube Service Life-Cycle

Andreas Metzger*, Eric Schmieders*, Cinzia Cappiello†, Elisabetta Di Nitto‡,
Raman Kazhamiakin‡, Barbara Pernici‡, Marco Pistore‡

*Paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen, 45127 Essen, Germany
Email: {andreas.metzger, eric.schmieders}@paluno.uni-due.de
‡FBK-Irste
Via Sommarive 18, 38050, Trento, Italy
Email: {raman, pistore}@fbk.eu
†Politecnico di Milano, DEI
Piazza Leonardo da Vinci, 32, 20133 Milano, Italy
Email: {dinitto, cappiell, pernici}@elet.polimi.it

Abstract—Service-oriented applications are deployed in highly dynamic and distributed settings. Therefore, such applications are often equipped with adaptation capabilities to react to critical issues during their operation, such as failures or unexpected changes of third party services or to context changes. In this paper, we discuss shortcomings of current solutions for adaptive service-oriented applications. To address those shortcomings, we introduce techniques that can be utilized to build and evolve proactive applications. Those techniques have been developed in S-Cube, the European network of Excellence on Software Services and Systems. Proactive adaptation capabilities are considered particularly promising, as they can prevent costly compensation and repair activities. Using those techniques in an integrated way is described along the phases of the service life-cycle. We use a running example to illustrate the shortcomings of current solutions for self-adaptation and to demonstrate the benefits of the S-Cube techniques.

I. INTRODUCTION

Service-orientation is increasingly adopted as a paradigm for building highly dynamic, distributed and adaptive software systems, called service-oriented (or service-based) systems. This paradigm implies a fundamental change to how software is developed, deployed, and maintained [1]: A service-based system cannot be specified and realized completely in advance (i.e., during design-time) due to the incomplete knowledge about the interacting parties (e.g., third party service providers) as well as the system’s context and communication infrastructure [2]. Thus, compared to traditional software engineering, much more decisions need to be taken during the operation of the service-oriented system (i.e., after it has been deployed). For instance, those systems will need to react to failures of their constituent services (e.g., if a service provider fails to adhere to its contract) to ensure that they maintain their expected functionality and quality.

In such a dynamic setting, evolution and adaptation methods and tools become key to enable those systems to respond to changing conditions. In accordance with the terminology defined by the S-Cube Network of Excellence [3], this paper differentiates between evolution and adaptation as follows: Evolution is considered as the modification of the system’s requirements, specification, models, etc. during design time (also known as maintenance). In contrast, adaptation is considered as the modification of a specific instance of a service-based system during operation. In the current paper we focus on adaptation needed due to some malfunctioning of the system. While the general adaptation due to context changes could also be supported by the proposed techniques, this is not discussed in the present paper.

A. Problem Statement and Related Work

Adaptive systems automatically and dynamically adapt to changing conditions. The aim of adaptation (aka. “self-adaptation”) is to reduce the need of human intervention as far as possible. While the behavior of a non-adaptive system is only controlled by user input, adaptive systems consider additional information about the application and its context (e.g., failures of constituent services or different network connectivity). Thus, in order to realize self-adaptive behavior, methods and tools that realize control loops are established that collect details from the application and its context (e.g., by exploiting monitoring mechanisms) and decide and act accordingly [4].

So far, the major work on adaptation has been centered around reactive adaptation capabilities based on monitoring [5]. This means that adaptation is performed after a deviation or critical change has occurred. Such a reactive adaptation based on monitoring, however, has at least the following two important shortcomings (cf. [6], [7] and [8]):

- It can take time before problems in a service-based system lead to monitoring events that ultimately trigger the required adaptation. One key trigger for an adaptation should be the case when the service-based system deviates from its requirements (such as expected response time for example). If only those requirements are monitored (e.g., see [9]), the monitoring events might arrive so late that an adaptation of the SBA is not possible...
anymore. For instance, the system could have already terminated in an inconsistent state, or the system has already taken more time than required by the expected response time.

- Reactive adaptation can become very costly, especially when compensation or rollback actions need to be performed. As an example, when using stateful (aka. conversational) services [10], the state of the failed service might need to be transferred to an alternative service.

Of course, one can monitor the individual services of an SBA and trigger an adaptation as soon as the service has failed, i.e., violated its contract [11]. However, when using those techniques it remains unclear whether the failure of this service could lead to a violation of the SBA’s requirements. This means that there may be situations in which the SBA is adapted although it would not have been necessary, because the requirements might still have been met. Consider the following simple example: Although a service might have shown a slower response time as (contractually) expected, prior service invocations (along the workflow) might have been fast enough to compensate the slower response of that service.

Such unnecessary (or “false positive”) adaptations have the following shortcomings [6]:

- Unnecessary adaptations can lead to additional costs and effort that could be avoided. For instance, additional activities such as Service Level Agreement (SLA) negotiation for the alternative services might have to be performed, or the adaptation can lead to a more costly operation of the SBA, e.g., if a seemingly unreliable but cheap service is replaced by a more costly one.
- Unnecessary adaptations could be faulty (e.g., if the new service has bugs), consequently leading to severe problems.

In summary, one key problem that needs to be solved to enable proactive adaptation is to determine whether the service-based application, during its future operation, might deviate from its requirements.

B. Contribution of Paper

This paper describes techniques developed in the S-Cube network of excellence to determine deviations from requirements based on monitored failures. Previous publications (such as [6], [7], [8], [9], [13]) have discussed proactive adaptation techniques mainly in isolation and confined to individual phases of the service life-cycle. A first, more integrated view on adaptation has been presented in [14]. However, the focus was on reactive adaptation and on the design time activities needed to build adaptive service-systems. In contrast, in this paper, we demonstrate how the techniques for determining pro-active adaptation play together across the various life-cycle phases and how they can be jointly applied in a meaningful way. As a basis for our discussions, we employ the S-Cube service life-cycle model [15], [14], [16], [17]. In contrast to more traditional life-cycle models, this model considers the specifics of service-based systems, particularly concerning evolution and adaptation.

The remainder of the paper is structured as follows: In Section II, the S-Cube service life-cycle model is introduced. In Section IV, the S-Cube techniques that jointly allow building proactive service-based systems are discussed, differentiating between activities that are done during design-time and activities that are done during the operation phase (run-time). This discussion is illustrated by an example from the eGovernment domain, which is introduced in Section III.

II. THE S-CUBE SERVICE LIFE-CYCLE MODEL

The life-cycle models for SBAs that have been presented in the literature (examples include SLDC, RUP for SOA, SOMA, and SOAD, cf. [14] and [18]) are mainly focused on the phases that precede the release of software and, even in the cases in which they focus on the operation phases, they usually do not consider the possibility for SBAs to adapt dynamically to new situations, contexts, requirement needs, service faults, etc.

Specifically, the following aspects have not yet been considered in those life-cycle models:

- **Requirements elicitation and design for adaptation**: The requirements engineering phase includes the elicitation and documentation of the systems functional and quality requirements. In the dynamic setting of SBAs, not only the requirements towards the actual application logic need to be analyzed, designed, and developed, but also the context in which the system is executed needs to be understood [1]. Context changes can necessitate the adaptation of the SBA, for instance if the SLA of a third party service is violated. During design, the capabilities to observe, modify and change the SBA during run-time need to be devised.

- **Extended operation phase**: The operation phase is not only responsible for merely executing and monitoring the application, but it also requires identifying the need for an adaptation of the system as well as the where and how to enact such an adaptation [1].

- **Continuous quality assurance**: Quality assurance has an impact on all aspects of the life-cycle. Therefore, the quality characteristics that are to be assessed and ensured must be identified starting from the requirement analysis phases. Due to open nature, and dynamic contexts in which SBAs operate, quality properties that have a lifelong validity need to be “continuously” asserted [19]. For instance, in the case of third party services, there is no guarantee that a service implementation eventually fulfills the contract promised (e.g., stipulated by an SLA), or it is usually not possible during design-time to model and thus assess the behavior of the underlying distributed infrastructure (such as the Internet).

The service life-cycle model envisioned by the S-Cube network aims at incorporating those aspects. The S-Cube service life-cycle model [15], [14], [16], [17] relies on two development and adaptation loops, which can be executed in an incremental and iterative fashion:

1http://www.s-cube-network.eu/
The development and evolution loop (see right hand side of Figure 1) addresses the classical development and deployment life-cycle phases, including requirements and design, construction and operations and management (see Section II-A).

- The operation and adaptation loop (see left hand side of Figure 1) extends the classical life-cycle by explicitly defining phases for addressing changes and adaptations during the operation of service-based applications (see Section II-B).

The requirements engineering phase, the functional and quality requirements for the SBA are elicited and documented. The specifics of SBAs make the requirements engineering phase particularly relevant. This is related to the highly dynamic nature of SBAs and to the necessity to guarantee the continuous adaptability and the evolvability of these applications. Indeed, in a context where the application is in continuous evolution and is characterized by very blurred boundaries, the study of those requirements that exist a priori in the organizational and business setting, and that are hence largely independent from the solution, becomes very important.

Design. During the design phase, the activities and the control flow of the application are specified. In the service-oriented case, this usually means that a workflow is specified using languages such as BPEL. Together with the definition of the workflow, candidate services are identified that can provide the functionality and quality to fulfill the requirements of the SBA. This means that those services that cover, at least partially, the expected functionality and quality of service are identified. This is supported by service matchmaking techniques, such as the ones presented in [20]. A further task in this phase is to define adaptation strategies and mechanisms which enable the application to react to adaptation needs (cf. [14]).

Construction. After the design phase, the construction of the system can start. Especially, it has to be taken into account that SBAs are obtained by the integration and coordination of services from different providers. Specifically, this means that for establishing the desired end-to-end quality of those SBAs, contracts between the service providers and the service consumers on quality aspects of services have to be established. Typically, this requires some form of SLA negotiation and agreement. Following [20], this means that for each service, the best quality of service level for the available budget is negotiated with the providers of the candidate services that have been identified in the previous phase.

Deployment and Provisioning. The deployment and provisioning phase comprises all the activities needed to make the SBA available to its users. It should be noted that an SBA can itself be offered as a service.

B. Operation and Adaptation Cycle

Operation and Management. This phase specifies all the activities needed for operating and managing an SBA. The literature also uses the term governance to mean all activities that govern the correct execution of SBAs (and their constituent services) by ensuring that they provide the expected functionality and level of quality during operation. In this setting, the identification of problems in the SBA (e.g., failures of constituent services) and of changes in its context play a fundamental role. This identification is obtained by means of monitoring mechanism and, more generally, by exploiting techniques for run-time quality assurance (such as online testing or run-time verification). Together, those mechanisms and techniques are able to detect failures or critical conditions.

Identify Adaptation Need. Some failures or critical conditions become triggers for the SBA to leave “normal” operation and enter the adaptation or evolution cycle. The adaptation cycle is responsible for deciding whether the SBA needs to be adapted in order to maintain its expected functionality and quality (i.e., to meet its requirements). This is an important decision as it might well be that despite a failure of a service, the end-to-end quality of the SBA is not affected and hence there is no need to react to that situation. Such decisions may be made automatically, or it may require human intervention (end user, system integrator, application manager). Moreover, such decisions may be made in a reactive way, when the problem has already occurred, or in a proactive way, where a potential, future problem could be avoided. It should be noted that the decision could also be that there should be an evolution of the system rather than an adaptation, thereby entering the “development and evolution” cycle.

Identify Adaptation Strategy. When the adaptation needs are understood, the corresponding adaptation strategies are identified and selected. Possible types of adaptation strategies include service substitution, SLA re-negotiation, SBA re-configuration or service re-composition. It could also happen that several adaptation strategies are able to satisfy a specific adaptation need. The selection of the strategy and its instantiation (e.g., which service to use as a substitute or which re-configuration to perform) may be automatic if either the SBA or the execution platform decide the action to perform, or it can be done by (the help of) a human operator. Specifically, two questions need to be answered: ”what to adapt?” and ”how to adapt?”.
Enact Adaptation. After the choice of the adaptation strategy, the adaptation mechanisms are used to enact the adaptation. For example, service substitution, re-configuration or re-composition may be obtained using automated service discovery and dynamic binding mechanisms, while re-composition may be achieved using existing automated service composition techniques. Depending on the situation, such an adaptation can be done manually (e.g., by a human operator), semi-automatically or fully-automatically.

III. APPLICATION SCENARIO

In this section, an example workflow is introduced in order to illustrate the problems as well as the solution that will be presented in Section IV. The workflow specifies an eGovernment SBA that allows citizens to pay parking tickets online, thereby saving effort and costs (see [21] for a description of the eGovernment application domain as defined in S-Cube).

A. Workflow

The workflow as well as the service composition of the eGovernment application are depicted in Figure 2 as an extended activity diagram. The gray boxes denote concrete services that can be composed to an eGovernment application. In the example, each service is provided by a third party, being it an external organization or a different unit of the governmental organization. Solid connections between workflow actions and services denote the bindings established at deployment time. Dashed connections denote possible alternative services (from a different provider). In addition, the diagram is annotated with information about the negotiated response times (which could be stipulated by means of SLAs).

Let us assume that the overall workflow is expected to have a response time of at most 1250 ms. This quality requirement can be satisfied by the bound services, provided that they meet their negotiated maximum response times (as, altogether, the maximum response times along the longest path add up to 1200 ms).

In the following subsections we use this example to illustrate the shortcomings of reactive adaptation, which have been introduced in Section I-A. We assume that the ePay service of the example workflow fails during runtime, i.e., takes longer than the negotiated maximum response time.

B. Scenario A: Requirements Monitoring

As mentioned in Section I-A there are approaches which are restricted to monitoring of requirements. In that case monitoring events might arrive so late that an adaptation of the SBA is not possible anymore. In our example, the ePay service invoked by Make Payment might take 650 ms to respond instead of the negotiated maximum response time of 400 ms (see Scenario A in Figure 3).

Due to the fact that only the requirement (maximum response time of 1250 ms) is monitored, this failure is not registered until after Sign has been invoked. As a consequence the mechanism was not able to prevent the deviation from the requirements, even though the failure has occurred much earlier (see Δ in Figure 3).

C. Scenario B: Service Monitoring

Referring to Section I-A, approaches that monitor individual services exist. However, in such setting it remains unclear whether the failure of a single service leads to a violation of the SBA's requirements. In the example, let us assume that instead of 400 ms the ePay service invocation takes 450 ms (see Scenario B in Figure 4).

This failure is observed by means of monitoring and leads to an adaptation of the SBA. However, as obvious in the figure, the overall response time would have still matched the required response time even if no adaptation would have been performed. Thus, in this case an adaptation was triggered although it was not necessary.

In the next section we will present techniques that enable a more proactive approach to addresses the above shortcomings.
IV. TOWARDS PROACTIVE ADAPTATION ALONG THE LIFE-CYCLE

This section describes techniques developed in S-Cube for enabling proactive adaptation. The description is organized along the phases of the life-cycle model from Section II. In order to illustrate the techniques, we refer to the example SBA and scenarios presented in Section III.

As explained in Section I-A, adaptive SBAs automatically and dynamically adapt to changing conditions and changes of service functionality and quality. To enable such an automatic adaptation, the relevant artifacts, as well as the properties of the SBAs and their context need to be formalized to make them amenable to automated checks and decisions. In the remainder of this section, we thus introduce concrete formalization approaches, as well as techniques that build on this formalization.

A. Requirements Engineering

To automatically assess whether the application deviates from its requirements during operation and thus trigger an adaptation, functional and non-functional requirements need to be collected and formally expressed. We propose to formalize the requirements already in the requirements engineering phase, as this also facilitates an early validation of the requirements, e.g., by means of formal consistency checks (cf. [22]), and hence reduces the risk of expensive corrections in later phases.

S-Cube has developed various approaches to formalize requirements (depending on the actual SBA type). For instance ALBERT is a specification language based on temporal logics (presented in [?]). ALBERT is used to encode functional and quality attributes. In addition, the S-Cube Quality Meta Model (QMM) has been defined, which provides a set of key concepts for expressing quality requirements and constraints (see [23]).

To express the requirements for monitoring, an integrated monitoring framework and the corresponding specification language has been provided (see [24] and [25]) in the scope of S-Cube. The framework integrates the capabilities of two monitoring platforms: Dynamo [11] and ASTRO [9]. On the one hand, the language enables the specification of complex point-wise properties over service composition execution (e.g., pre- and post-conditions on service calls), taking into account current and historical values of the process variables, complex constraints, and event external properties. On the other hand, simple events and point-wise properties may be aggregated into complex behavioral expressions, also taking into account temporal and statistical information necessary for capturing non-functional requirements. While the latter capability is very close to the approach used by ALBERT, the notation allows for expressing properties over classes of processes rather than over single instances. This capability may be very important in order to trigger “evolution” of the workflow, when the problem applies to the whole SBA model rather than to a single SBA instance.

Example: In our example from Section III, we need to formalize the required response time \( r_{\text{perf}} \) of the eGovernment application. \( r_{\text{perf}} \) is an element of the given set of requirements \( R_{\text{eGov}} \) against the eGovernment application:

\[
\forall r_{\text{perf}} \in R_{\text{eGov}}
\]

\( r_{\text{perf}} \) demands the response time of the eGovernment application to be at most 1250 milliseconds. Due to the capability of ALBERT to express the dependencies of monitoring data along an executed path, we choose this language to specify the requirement (\( r_{\text{perf}} \)) as follows:

\[
\begin{align*}
\text{r}_{\text{perf}} & := \text{onEvent}(\text{start},"Identify Parking Ticket") \\
& \quad \rightarrow \text{Within}(\text{onEvent}(\text{end},"Send eMail"), 1250)
\end{align*}
\]

The \text{onEvent} operator evaluates to true if the activity specified in the second argument performs the state change denoted in the first argument. The \text{Within} operator evaluates to true if its first argument evaluates to true within the amount of milliseconds specified in its second argument.

B. Design

Similarly to the requirements, the workflow of the SBA needs to be formalized to support automated checks. Following the same reasoning as in the requirements engineering phase (see above), we suggest to formalize the workflow during the design phase already in order to reduce the risk of later corrections. As presented in [8] the checks can be performed by using Model Checking techniques. In S-Cube the use of BOGOR has been proposed [?7] to assess whether the specified SBA satisfies the requirements. We thus formalize the workflow using the input language of the model checker, in this case BIR (Bogor Input Representation).

Example: In order to use the BOGOR Model Checker (as proposed in [?]) we specify the eGovernment Workflow by
using BIR. The resulting specification $S_{eGov}$ (see Listing 1 in the appendix) can be directly executed and analyzed by BOGOR.

C. Realization

During the realization phase, the quality levels (aka. service level objectives) that have been negotiated and agreed upon with the service providers (see Section II), are formalized.

Following the proposal in [8], we treat those quality levels as assumptions ($A$) about the SBA’s context. Due to the lack of control of third-party services, those quality levels could be violated during the operation of the SBA (see Section I). To formalize $A$, we can use one of the quality formalization approaches as used during the requirements engineering phase.

For checking the violation of the assumptions during the operation of the SBA, monitoring mechanisms are implemented that collect the relevant data (cf. [24], [25] and [27]). This is equivalent to collecting the monitoring data in the reactive case of adaptation (cf. Section III).

Example: According to their SLAs (see Figure 2) ALBERT is used to formalize the five assumed response times. The set of assumptions $A_{eGov}$ for the parking ticket SBA is defined as

\[ A_{eGov} := \{ a_{DeptATicketHandler}, a_{ePay}, a_{DeptCTicketHandler}, a_{eSign}, a_{Yahoo} \} \]

The assumption $a_{ePay}$, related to the $ePay$ service invocation, is formalized as follows:

\[ a_{ePay} := \text{onEvent} (\text{start}, "\text{Make Payment"}) \rightarrow \text{Within} (\text{onEvent} (\text{end}, "\text{Make Payment"}), 400) \]

D. Deployment

Before deploying the SBA, it is checked whether the workflow specification ($S$), under the given assumptions ($A$), satisfies the requirements ($R$):

\[ S, A \models R \]

This check ensures that the initial composition – the workflow and the services – satisfy the requirements. If this is not the case, the phases of the evolution loop (cf. life-cycle in Section II-A) are executed again in order to redesign the application, e.g., to bind faster services. If the SBA is successfully verified against the requirements the SBA is deployed.

Example: In our example $S_{eGov}$ and $A_{eGov}$ satisfy $R_{eGov}$. In consequence the SBA is deployed.

E. Operation and Management

This phase comprises the execution and the monitoring of the individual services of the deployed SBA.

Monitoring is supported by monitoring frameworks, such as Dynamo (presented in [25]). During runtime, the monitoring framework continuously assesses whether the monitoring data $M$ satisfies the formalized assumptions $A$ about the services:

\[ M \models A \]

If a violation occurs, the SBA enters the adaptation loop (cf. Section II-B). The relevant activities are described below (Sections IV-F, IV-G, and IV-H).

Example: After finishing the SBA deployment, the eGovernment application is executed. Let us assume that the first activity, which invokes the DeptATicketHandler service, lasts 90 milliseconds. The measured response time of the DeptATicketHandler call is stored as monitored data $m_{DeptATicketHandler}$. $m_{DeptATicketHandler}$ satisfies the assumption that the service responds within 100 milliseconds ($a_{DeptATicketHandler}$). In the next step $ePay$ is invoked. Let us assume, that the invocation of $ePay$ is slower than expected. This is the same situation as described in Scenarios A and B (see Section III). Instead of 400 milliseconds as expected, the $ePay$ invocation takes 450 milliseconds (cf. Scenario B). Hence, the monitoring data of the second service invocation $m_{ePay}$ doesn’t satisfy the corresponding assumption $a_{ePay}$:

\[ m_{ePay} \not\models a_{ePay} \]

Due to this violation the phases of the adaptation loop are entered.

F. Identify Adaptation Needs

In this phase it is checked, whether the requirements are still satisfied, although the assumptions have been violated (cf. [8]). For example it might be the case that a slower response time of one service is compensated by a faster response time of a previous service, and consequently no adaptation is required.

When the check is performed, there usually are services which have not been invoked. Only when the workflow is finished, all services have been invoked. This means, that there is no monitoring data available for the not yet invoked services. For those not yet invoked services we continue to use their assumptions in the checks, i.e. we use a subset $A' \subseteq A$. Next, it is checked, whether the workflow specification $S$, the monitored data $M$ and the assumptions in $A'$ satisfy the given requirements $R$.

\[ S, M, A' \models R \]

If $R$ is satisfied, then the workflow execution is continued. If $R$ is not satisfied, the SBA must be adapted.

Example: To illustrate that the presented S-Cube approach is adequate to address the shortcomings from III-B and III-C, we compare the S-Cube approach with the requirements monitoring approach presented in Section III-B (Scenario A) and the sequence monitoring approach presented in Section III-C (Scenario B). It is checked, whether there is a deviation from the requirements, as this could indicate that an adaptation is necessary. This check also covers cases with larger delays, e.g., 500 milliseconds in Scenario A.

The approach presented in Scenario A (see Section III-B) does not observe failures at the moment when they occur –
as depicted with $\Delta$ in Figure 3. The S-Cube approach does not have this shortcoming. The continuous monitoring of the service behavior observes failures as soon as a problem occurs. Based upon such an observation, the SBA requirements are immediately checked. This provides the system with the opportunity to adapt itself to prevent the predicted requirements deviation from occurring. Of course the ability of the system to proactively adapt depends on the time available for such actions. Typically, if a failure of a service is observed more at the beginning of the workflow, more time remains to adapt the remainder of the workflow accordingly.

In order to determine such requirements violations, Model Checking techniques are used. The workflow specification ($S_1$), the monitoring data ($m_{\text{Dept ATicketHandler}}$ and $m_{\text{ePay}}$) together with the assumptions of the outstanding service invocations ($a_{\text{Dept ATicketHandler}}, a_{\text{eSign}}$ and $a_{\text{Yahoo}}$) are checked against the requirement $r_{\text{perf}}$. The expected overall runtime is 1450 milliseconds which exceeds the 1250 milliseconds demanded in $r_{\text{perf}}$. Hence, the requirement $r_{\text{perf}}$ is not satisfied. This result is considered as an identified adaptation need. Subsequent to this check, the adaptation can be performed proactively, before the requirement is actually violated (i.e., before the system in operation deviates from its expected requirements).

The approach presented in Scenario B (see Section III-C) is not able to determine, whether a failure of a single service leads to a violation of the SBAs requirements. Each time a service fails, the SBA adapts immediately. The S-Cube approach presented in this paper allows adapting only in cases when critical failures occur, thereby avoiding unnecessary adaptations. The same check as described above assesses that the expected overall runtime does not exceed 1250 milliseconds. The requirement $r_{\text{perf}}$ is still satisfied and thus no adaptation trigger is needed. Thereby an unnecessary adaptations is prevented, which would have been performed in Scenario B.

G. Decide on Adaptation / Identify adaptation strategy

When the need for adapting an SBA is detected, the next step is to identify and apply an appropriate adaptation strategy among the ones that are available for the considered applications. Depending on the application, the adaptation strategies may range from service re-execution, over replacement of a single service or of the process fragment, over re-negotiation of quality properties, to changes in underlying infrastructure, etc. Note that the adaptation strategies should be designed with the application since some of them require the adoption of specific infrastructure or the implementation of additional components.

Typically, the adaptation strategy is associated with a specific critical situation or a problem at design time. This association may be done either implicitly or explicitly. In the former case, the mechanisms for choosing one action or another are “hard-coded” in some decision mechanisms. A typical scenario is the replacement of a service that violates the SLA or a SBA requirement with a new one, with appropriate and most suitable characteristics. Based on the selection criteria (e.g., optimization of a quality function, adherence to application constraints), the appropriate decision mechanism may choose one service or another. In the scope of the S-Cube project, several approaches follow this vision. For example, in [26] the replacement policies realize such a decision mechanism and define the association between various types of changes (service failure, changes in service properties and models, appearance of new services, and changes in the context and requirements) and the service selection. In [27], the decision on the adaptation strategy is based on the quality factors of the SBA that should be improved. Those factors are identified through the analysis of the dependency tree that capture the relation between simple quality factors and SBA requirements. At design time, the adaptation action is assigned to the quality factors that it influences either positively or negatively. The selection of the adaptation strategy is based on the need to improve quality factors that are critical for the requirement, while trying to minimize the negative effect on the other factors. In our scenario, the requirement would need to improve the performance of the last service, and the service replacement would be proposed such that the new service has better performance, while having smaller cost with respect to alternatives.

The definition of the adaptation strategy may be also explicitly assigned to the critical situation. For example in [28] the adaptation strategy is represented in the WS-ReL, a notation for specifying and integrating recovery actions in service composition. Therefore, the adaptation is defined as a rule, where in the left hand side a critical situation is defined (as a formal requirement to be monitored) and in the right hand side a set of actions to be applied. The possible actions include re-execution of a service invocation, replacement of a service or a provider (partner link), ignoring the failure or halting the execution, executing an extra process fragment, or rolling back to a safe point. Simple actions may be joined into a complex strategy by defining a control flow over actions, like “try action A else try action B and action C”. These rules are evaluated and applied by the underlying adaptation engine.

Adaptation can also be based on the causes of failures. This is particularly helpful when invoked services are stateful, and their invocation modifies the state of the service, such as in transactional services. For processes involving transactional services, if a diagnosis mechanism is available, such as in [7], the adaptation strategy can depend on the cause of the failure and its implications on the processes. This might imply an adaptation strategy involving one or more services in the process which must be dynamically generated.

H. Enact Adaptation

To enact adaptation actions, the SBA or its execution platform should be appropriately instrumented. A typical approach for realizing adaptation mechanisms for SBAs implemented as executable (BPEL) processes is to instrument the process execution engine. Such instrumentation is done via Aspect-Oriented Programming techniques, as the adaptation activities
orchestrations. Services into service compositions that are realized as BPEL-oriented techniques are adopted in order to dynamically bind to realize the specific strategy. Similarly, in [29], where aspect-adaptation rules, and calls to the process engine infrastructure are treated as a cross-cutting concern. Using this approach, the join points allow for injecting the adaptation logic in order to respond to failures or unexpected changes of third party services. In S-Cube, we are currently striving to push the envelope towards proactive adaptation even further. In addition to determining the need for adapting the service-based application based on actual failures of the application’s constituent services, we investigate the applicability of online testing for predicting the quality of those services (e.g., see [6], [7]). Combined with the approaches introduced in this paper, this means that critical problems could be observed even earlier, thus enabling a broader range of adaptation and evolution strategies. For instance, in our running example we can only react to the violation of the response time of a constituent service by ensuring that the remainder of the workflow executes faster. However, if the quality prediction techniques forecast a violation of the expected response time of a specific service, this very service can be replaced before it is invoked in the context of the service-based application.

Another challenging problem that will be addressed is how to make the techniques robust against other kinds of “false-positives”. Currently our techniques define the assumptions about a service execution to be the upper limits of the quality properties as stated in the SLAs. As a consequence, it might happen that the proactive techniques predict a performance requirements violation based on a failure of a service despite the fact that the remaining service invocations of the workflow might have executed much faster than stated in the SLAs and thus compensating for this failure. We thus will investigate in how far past monitoring data could be used to better define the assumptions that can be made about the quality properties of a service.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their helpful and constructive comments.

The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube). For further information please visit http://www.s-cube-network.eu/.

REFERENCES


APPENDIX

Listing 1. Workflow Specification $S_{eGov}$

```plaintext
system ParkingTicketSpec {
  Action identifyParkingTicket;
  Action makePayment;
  Action updateParkingTicketRecord;
  Action sign;
  Action sendEMail;
  int sumMaxResponseTime := 0;
  record Action {
    string serviceName;
    int maxResponseTime;
    boolean serviceInvoked;
  }
  active thread MAIN () {
    init();
    checkWorkflow();
    checkRequirements();
  }
  function init () {
    identifyParkingTicket := createAction("DeptATicketHandler", 100);
    makePayment := createAction("ePay", 400);
    updateParkingTicketRecord :=
      createAction("DeptCTicketHandler", 500);
    sign := createAction("eSign", 100);
    sendEMail := createAction("Yahoo", 100);
  }
  function checkWorkflow() {
    executeAction(identifyParkingTicket);
    choose
      do skip;
      do atomic
        executeAction(makePayment);
        executeAction(updateParkingTicketRecord);
      choose
        do skip;
        do atomic
          executeAction(sign);
        do skip;
        end
      end
    end
  }
  function checkRequirements() {
    assert sumMaxResponseTime <= 1250;
  }
  function executeAction(Action action) {
    sumMaxResponseTime :=
      sumMaxResponseTime +
      action.maxResponseTime;
    action.serviceInvoked := true;
  }
  function createAction(
    string serviceName,
    int maxResponseTime) returns Action{
    Action action;
    action := new Action;
    action.serviceName := serviceName;
    action.maxResponseTime := maxResponseTime;
    action.serviceInvoked := false;
  }
```


return action;
A Process Reference Model for Developing Adaptable Service-Based Applications

Stephen Lane¹, Antonio Bucchiarone², Ita Richardson¹

February 17, 2011

¹Lero, the Irish Software Engineering Research Centre, University of Limerick, Ireland
²Fondazione Bruno Kessler, Via Santa Croce 77 38100 Trento, Italy

Abstract

Context: The loose coupling of services and Service-Based Applications (SBAs) have made them the ideal platform for context-based run-time adaptation. There has been a lot of research into implementation techniques for adapting SBAs, without much effort focused on the software process required to guide the adaptation.

Objective: This paper aims to bridge that gap by providing a empirically grounded software process model that can be used by software practitioners who want to build adaptable SBAs. The process model will focus on adaptation specific issue meaning that it will not be a complete SBA development life-cycle.

Method: The process model presented in this paper is based on data collected through interviews with 7 practitioners occupying various roles within 7 different companies, and a 3 interview case study with an additional company. The data was analyzed using best practice qualitative data analysis techniques resulting in a set of activities, tasks, stakeholders and artifacts that were used to construct the process model.

Results: The outcome of the data analysis process was a process model with 9 sets of adaptation process attributes that can be used in conjunction with an organisation’s existing development life-cycle or another reference life-cycle.

Conclusion: The process model developed in this paper provides a solid reference for practitioners who are planning to develop adaptable SBAs. It have advantages over similar approaches in that it focuses on software process rather than the specific adaptation mechanism implementation techniques.

Contents

1 Introduction 3
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Objectives</td>
<td>4</td>
</tr>
<tr>
<td>1.2</td>
<td>Taxonomy</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Background</td>
<td>6</td>
</tr>
<tr>
<td>2.1</td>
<td>A life-cycle to develop Adaptable SBA</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Design Guidelines for Adaptation</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Research Method</td>
<td>8</td>
</tr>
<tr>
<td>3.1</td>
<td>Construction of the FoR Model</td>
<td>8</td>
</tr>
<tr>
<td>3.2</td>
<td>Model V1</td>
<td>10</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Expert Opinion Survey</td>
<td>10</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Case Study</td>
<td>11</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Data Analysis</td>
<td>12</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Evaluation</td>
<td>14</td>
</tr>
<tr>
<td>4</td>
<td>FoR Model</td>
<td>14</td>
</tr>
<tr>
<td>4.1</td>
<td>FoR Model Description</td>
<td>16</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Requirements Engineering</td>
<td>16</td>
</tr>
<tr>
<td>4.1.2</td>
<td>Design</td>
<td>17</td>
</tr>
<tr>
<td>4.1.3</td>
<td>Construction</td>
<td>17</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Deployment and Provisioning</td>
<td>17</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Operation and Management</td>
<td>18</td>
</tr>
<tr>
<td>4.1.6</td>
<td>Identify Adaptation Triggers</td>
<td>18</td>
</tr>
<tr>
<td>4.1.7</td>
<td>Identify Adaptation Strategies</td>
<td>19</td>
</tr>
<tr>
<td>4.1.8</td>
<td>Enact Adaptation</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>Model V1</td>
<td>19</td>
</tr>
<tr>
<td>5.1</td>
<td>Processes and Activities</td>
<td>20</td>
</tr>
<tr>
<td>5.2</td>
<td>Detailed Process Attributes</td>
<td>22</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Requirements Engineering</td>
<td>22</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Design</td>
<td>22</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Construction</td>
<td>23</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Testing</td>
<td>24</td>
</tr>
<tr>
<td>5.2.5</td>
<td>Deployment</td>
<td>24</td>
</tr>
<tr>
<td>5.2.6</td>
<td>Operation and Management</td>
<td>25</td>
</tr>
<tr>
<td>5.2.7</td>
<td>Identify Adaptation Triggers</td>
<td>25</td>
</tr>
<tr>
<td>5.2.8</td>
<td>Select Adaptation Strategies</td>
<td>26</td>
</tr>
<tr>
<td>5.2.9</td>
<td>Enact adaptation</td>
<td>26</td>
</tr>
</tbody>
</table>
1 Introduction

The Service-Oriented Computing (SOC) paradigm [1] is centered around the publication and consumption of loosely coupled computational elements known as services. These software services are often owned and controlled by third parties rather than the application developers who consume them. A key benefit of SOC is the ability to rapidly compose distributed software services into useful Service-Based Applications (SBAs). These applications are able to offer complex and flexible functionalities in widely distributed environments by composing different types of services. Such services are often not under the control of systems developers, but they are simply exploited to obtain a specific functionality.

The possibility of rapidly composing SBAs opens up new opportunities for conducting ad-hoc business transactions. However, with this new freedom there are also new challenges, for example, when services are owned and controlled by third parties their reliability or availability are not guaranteed. Adaptable SBAs change their behavior, reconfigure their structure and evolve over time reacting to changes in the operating conditions, so as to always meet users’ expectations. This is fundamental since those systems live in distributed and mobile devices, such as mobile phones, PDAs and laptops which have frequently changing environments. Also, user goals and needs may change dynamically, and systems should adapt their functionalities accordingly, without intervention from technicians.

Modeling of the SBA logic is not the only problem to consider. The adaptability of SBAs also needs to be considered, adaptation needs to be considered during the requirements engineering, design and construction phases of SBA development. Similarly the
adaptation of an SBA needs to be monitored and controlled when these applications are in operation.

Many of the existing SBA development methodologies are based on the results carried out in the fields of classical software and system engineering and do not easily facilitate SBA adaptation [2][3][4]. Some of the reported SBA development approaches such as SOUP (Service Oriented Unified Process) [5], ASTRO [6] or an approach by Linner et al [7] do support some level of adaptation, however, lack sufficient process details. Lane and Richardson [8] carried out a systematic literature review of SBA development approaches, they identified 57 such approaches of which there were only 8 that specifically dealt with adaptation. Only 4 of the 8 approaches were concerned with adaptation of SBAs, the others were concerned with the adaptation of services. Each of these 4 approaches focused on the analysis and design processes without consideration for any other development or runtime processes.

All aforementioned approaches show interesting features, but even those that enable the definition of various adaptation strategies lack a coherent design approach to support designers in this complex task. Moreover, they tend to focus on the implementation specific details without considering what impact adaptation has on the rest of the development and operational life-cycle [9][10][11][12]. Finally, they also tend to focus on particular types of adaptation, such as adaptation due to requirements violations [13], or substitution services due to application constraints [14], so it is difficult to elicit generic adaptation specific processes from them. The solution proposed in [15] can be considered as a first step in this direction. The authors of this paper have proposed a process model for SBAs where adaptation is a first class concern. This process model provides a number of steps that, if implemented during the life-cycle of a SBA, will allow the SBA to adapt in response to changing runtime contexts.

1.1 Objectives

The primary objective of the paper is to develop an empirically grounded process model that can be used as a reference for service practitioners who want to develop adaptable SBAs. The process model will focus on activities related to adaptation only, meaning that it will have to be used in conjunction with some other SBA development life-cycle. In order for an SBA to be adaptable, certain adaptation mechanisms need to be put in place when the application is initially developed. These mechanisms then facilitate adaptation when the application is in operation.

There are two types of adaptation, static adaptation and dynamic adaptation [16]. With static adaptation, adaptation mechanisms are hard coded into the application at development time and the adaptation logic cannot be changed without recoding. Dynamic
adaptation on the other hand allows the adaptation logic to be modified or replaced during runtime without shutting the system down. Dynamic adaptation is more flexible than static adaptation but it requires some process to guide the manual intervention during runtime.

This research focuses on dynamic adaptation which, as previously mentioned, has a runtime element. Process details will be elicited from relevant SBA development and adaptation approaches as well as qualitative data gathered from interviews with industry practitioners. The validity of the SBA adaptation process model will be demonstrated through a comparison with a previously validated adaptation process model. This comparison will show that the process model contains the minimum set of activities required for runtime software adaptation. The processes and activities from the process model will also be mapped to an empirically based SBA development life-cycle. This will illustrate whether or not the model is suitable for use in a real world SBA development context.

1.2 Taxonomy

The terminology used in the areas of software process and service engineering research are often ambiguous and conflicting. In order to define what is meant by the more common software process and service engineering terms used in this paper a brief outline will be given for each. The definitions for the service engineering terms are based on the S-Cube knowledge model [17] while the software process terms are based on Derniame et al [18].

Service-Based Applications are composite applications composed of multiple software services. Software services or services as they are more well known are computational elements that expose their functionality over computer networks. Because services are self-contained and loosely coupled it is possible to build SBAs from unrelated services that may or may not be in the control of the SBA developers. An example SBA often referred to is the travel booking application, this is a travel reservation application composed of three third party services: a car rental, flight reservation and hotel reservation services. These three unrelated services are composed to provide a useful SBA, synonyms for SBAs include service-oriented systems or service-based systems.

Adaptable Service-Based Applications ASBAs are a specific type of SBA that adapt during operation by following built-in adaptation strategies. Adaptation of SBAs is facilitated by monitoring mechanisms that monitor changes in monitored properties. Adaptation may be desirable for many reasons such as changes in quality characteristics, change in business requirements or change in the cost of services being consumed. Adaptation may occur through the selection of alternative services from pre-defined lists or through discovery of new services from service directories.
**Software Process** Software Process refers to the set of processes and activities undertaken to develop software. This process is often measured so that it can be managed or improved upon. Some software developers follow predefined processes and best practices while others have a more ad-hoc software process. Nevertheless, whether defined or not, each software development project will follow a particular set of processes. Synonyms for software process include: software development process or software life-cycle.

**Process Models** are a documented representation of a set of real world processes, activities and their interrelations. A **Software Process Model** is specific type of process model for software development. Creating a Process Model for a software development process allows reasoning about the model so that it can be improved upon. Ideally a process model should perfectly articulate the real world processes and activities that they are attempting to describe.

**Process Reference Models** are best practice or exemplar **Process Models** that can be used as a reference by individuals or groups to improve on their own processes. A specific type of Process Reference Model is a **Software Process Reference Model** which is an exemplar process model used for developing software, for example, the Capability Maturity Model Integration (CMMI) and ISO-15504 document process reference models.

**Life-Cycle Models** in software engineering are **Process Models** that define the entire software development life-cycle. This is different from a **Process Model** which may only address particular parts of the development cycle.

**Life-Cycle Process** a software life-cycle process is a collection of software development activities that occur in sequence and make up one logical part of the software development life-cycle, for instance, “Requirements Engineering” is a **Software Life-Cycle Process** that may contain activities such as ”Elicit requirements” or ”Develop requirements specifications”.

## 2 Background

### 2.1 A life-cycle to develop Adaptable SBA

The life-cycle shown in Figure 2 highlights the typical design-time iteration cycle that leads to the explicit re-design of the application in order to adapt it to new needs. It also introduces a new iteration cycle at run-time that is undertaken in all cases in which the adaptation needs are addressed on-the-fly. The two cycles coexist and support each
other during the lifetime of the application. In particular, the design-time activities allow for evolution of the application, that is, for the introduction of permanent and, usually, important changes, while the run-time activities allow for temporary adaptation of the application to the specific circumstances that are occurring at a certain time.

2.2 Design Guidelines for Adaptation

The adaptation in SBA may be motivated by variety of needs. Such needs may concern the component services or the context of SBAs. We can identify:

**Changes in the service functionality** due to variation of the service interface (e.g., signatures, data types, semantics), variation of service interaction protocol (e.g., ordering of messages) or Application failures.

**Changes in the service quality** due to service availability, degrade of QoS parameters, violation of SLA or decrease of service reputation (e.g., black lists), etc.

**Changes in the service context** as a result of changes in the business context, changes in agile service networks, or new business regulations and policies.

**Changes in the computational context** such as different devices, protocols, and networks.

**Changes in the user context** such as different user groups and profiles, social environment, physical settings (e.g., location/time), and different user activities.

Some of these aspects may be interleaved. For example, if a user moves to a new location (i.e., change in the user context), new set of services may be available (i.e., change in the business context) with different bandwidth (i.e., change in the computational context). Each factor can be associated with a set of adaptation strategies that are suitable to re-align the application within the system and/or context requirements. In order to select the adaptation strategy to apply, it is necessary to consider that adaptation needs may be associated with other requirements that are important for designing and performing adaptation. In particular, the scope of the change, i.e., whether the change affects only a single running instance of the SBA or influences the whole model and, the impact of the change, i.e., the possibility of the application to accomplish its current task should be considered. Depending on these parameters different strategies may apply.

Among the adaptation strategies, it is possible to distinguish domain-independent or domain-dependent strategies. The former are applicable in almost every application context while the adoption of the latter is limited to specific execution environments. Table 1 defines the most common domain-independent adaptation strategies.
Table 1: Description of the most common domain-dependent adaptation strategies

<table>
<thead>
<tr>
<th>Adaptation Strategy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service substitution</td>
<td>Reconfiguration of the SBA with a dynamic substitution of the a service with another one</td>
</tr>
<tr>
<td>Re-execution</td>
<td>The possibility of going back in the process to a point defined as safe for redoing the same set of tasks or for performing an alternative path</td>
</tr>
<tr>
<td>(Re-)negotiation</td>
<td>Simple termination of the service used on the requester side and re-negotiation of the SLA properties to complex management on reconfiguration activities on the provider side</td>
</tr>
<tr>
<td>(Re-)composition</td>
<td>Reorganization and rearrangement of the control flow that links the different service components in the business application</td>
</tr>
<tr>
<td>Compensation</td>
<td>Definition of ad-hoc activities that can undo the effects of a process that fails to complete</td>
</tr>
<tr>
<td>Trigger evolution</td>
<td>Insertion of workflow exception able to activate the application evolution</td>
</tr>
<tr>
<td>Log/update adaptation information</td>
<td>Storage of all the information about the adaptation activities for different goals (e.g., service reputation, QoS analysis, outcome of adaptation)</td>
</tr>
<tr>
<td>Fail</td>
<td>The system reacts to the changes by storing the system status and causing the failure of the service and re-executing it</td>
</tr>
</tbody>
</table>

The identification of the most suitable strategy is supported by a reasoner that also bases its decisions on multiple criteria extracted from the current situation and from the knowledge obtained from previous adaptations and executions. After this selection, the enactment of the adaptation strategy is automatically or manually performed. The execution of all activities and phases in all runtime phases may be performed autonomously by SBAs or may involve active participation of the various human actors.

3 Research Method

There were two phases involved in the construction of the process model during this study. The first phase involved the development of a Frame of Reference (FoR) based on relevant literature. This was enhanced with empirical evidence gathered during the second phase of the study, resulting in Version 1 (V1) of the process model. The research design largely follows Ahlemann and Gastl’s [19] process for empirically grounded reference model construction.

3.1 Construction of the FoR Model

The FoR model was constructed by analysing existing publications that contain activities for adapting SBAs. Peer-reviewed as well as non-peer-reviewed publications were searched for these process details. Once relevant publications had been retrieved they were analysed for activities that could be used to construct a generic FoR model. Many of the activities identified were similar and at varying levels of abstraction. In order to eliminate duplication and to show the activities at the same level of abstraction, each activity was assigned a generic name. Similar activities were assigned the same generic name, resulting in a smaller list of unique generic activities. The “activity” abstraction level, as
defined in ISO/IEC 15288 [20], was chosen for the generic activity names. ISO/IEC 15288 defines activities as a group of related tasks that are required to achieve the outcomes of a process. Figure 1 [20] illustrates, using UML\(^1\) class diagram notation, the ISO/IEC 15288 process constructs and their interrelations. The figure shows that a Process can have zero or more Sub-Processes and one or more Activities. In turn, it shows that an Activity can have one or more tasks which can have zero or more Notes. It also shows that each process has a Name, a Purpose, and an Outcome.

The S-Cube life-cycle, illustrated in Figure 2 [21], was chosen as a starting point to develop the FoR. The S-Cube life-cycle is a high-level life-cycle model skeleton proposed for the development of adaptable SBAs. The generic adaptation activities identified were mapped to the appropriate processes of the S-Cube life-cycle resulting in the FoR model. This approach was taken in order to leverage the existing research which has been conducted by the S-Cube consortium. Additionally, the S-Cube life-cycle covers each aspect of the development cycle without being too detailed. This allows it to be easily modified and enhanced with activities reported in the service engineering literature. The FoR model is a high-level model and only uses the Process and Activity constructs as defined in ISO/IEC 15288.

\(^1\)www.uml.org
3.2 Model V1

V1 of the process model was constructed with data gathered from an expert-opinion survey and an interview-based case study.

3.2.1 Expert Opinion Survey

The first stage in the expert-opinion survey was the development of an interview guide for use during the survey interviews. The interview guide was divided in two parts: the first part deals with the participants backgrounds and their roles, while the second part focuses on the elicitation of adaptation activities. The second part of the guide asks participants to comment on the activities within the FoR model, developed in the first part of this study, and to provide feedback. The respondents were asked whether or not they agreed with the activities contained in the FoR, and what changes they would make. Then they were asked to describe the tasks required to complete these activities, the stakeholders involved and the artifacts produced by the activities.

The interviews conducted were semi-structured interviews [22]. Interview guides for this type of interview contain open-ended questions which allow interviewers to ask follow on questions when necessary. This flexibility allows the interviewer to ask the interviewees detailed questions about their specific areas of expertise. A pilot interview was conducted using this interview format. This results of the pilot showed that this approach struck a good balance between flexibility and structure. A strictly structured questioning approach would have prevented the interviewer from probing concepts that were not considered while developing the FoR. The interview guide used for the interviews is contained in Appendix B.

Once the interview guide was completed and piloted, the next step was to identify and
contact suitable interview partners. This proved to be one of the more challenging aspects of the study given the specialised nature of the process model being constructed. In order to participate, interview partners needed to be experts in Service-Oriented Computing (SOC) and be able to provide opinion, based on experience if possible, on how best to adapt SBAs. Suitable practitioners were identified through user groups on the Linkedin² professional network as well as SOC conference proceedings. After sending out invitations to over 100 practitioners there were 15 replies with 7 eventually committing to interviews. At the end of each interview participants were asked to identify others who may participate with some initial positive responses. However, the initial interview partners did not provide leads for additional interviews indicating that chain sampling can be ineffective for such a specific research project topic.

In advance of the interviews the interviewees were sent a copy of the interview guide as well as some background information on the study. This would allow them to familiarise themselves with the concepts and terminologies involved in the study. It would also allow them extra time to think about the adaptation activities that they encountered while developing SBAs. The interviews were conducted either on site or through the Skype™ Voice Over Internet Protocol (VOIP) service which allows the recording of interviews which is necessary for transcription. Once all of the interviews were complete, they were transcribed in preparation for data analysis.

3.2.2 Case Study

A single case study was also conducted in order to supplement the survey conducted during this inquiry. The case study was based on a typical company with expertise in developing SBAs. The aim of this typical case study [23] was to identify how a typical SBA development team approaches the adaptation of SBAs. The company where the case study was carried out, in order to protect its identity, will be referred to as SbaSoft.

Interviews were used in order to collect qualitative data during the case study. Three interviews were held in total with stakeholders involved in different aspects of the application development life-cycle: The Chief Technology Officer (CTO), a business analyst and a developer. The interviews for the case study were conducted on site at SbaSoft.

The interviews for the case study were based on a semi-structured interviews similar to that used in Section 3.2.1. However, rather than focusing on adaptation specific activities, the questions were aimed at documenting SbaSoft’s entire development life-cycle including adaptation specific activities. Their entire development life-cycle was documented in order to view their adaptation specific activities in context. After the interviews were complete they were transcribed so they could be analysed and provide input for V1 of the model.

²http://www.linkedin.com
3.2.3 Data Analysis

Once the data collection from the survey and case study was complete the interview transcripts were analysed using data analysis techniques as described in Miles and Huberman’s [24] expanded source book. The raw data was first open coded with descriptive codes being added to segments of the data. A starting list of codes was based on the FoR constructed in the first half of this study. Starting codes, as illustrated in Table 2, were created for activities, stakeholders and artifacts related to the FoR’s high-level processes. New codes were added and existing ones modified to accurately reflect the raw data as the coding proceeded. Participant background information was coded in chunks containing information about their current roles as well experience which provides context for their interview responses.

The coding was conducted with ulQda [25], a \LaTeX\ qualitative data analysis package. This package allows codes to be embedded within the raw data where they can be easily reviewed and edited in context. The package also allows the raw data to be typeset

Table 2: Starting Codes

<table>
<thead>
<tr>
<th>Category</th>
<th>Process</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>Requirements Engineering</td>
<td>activity!requirements engineering</td>
</tr>
<tr>
<td></td>
<td>Design</td>
<td>activity!design</td>
</tr>
<tr>
<td></td>
<td>Construction</td>
<td>activity!construction</td>
</tr>
<tr>
<td></td>
<td>Deployment and Provisioning</td>
<td>activity!deployment and provisioning</td>
</tr>
<tr>
<td></td>
<td>Operation and Management</td>
<td>activity!operation and management</td>
</tr>
<tr>
<td></td>
<td>Adaptation Triggers</td>
<td>activity!adaptation triggers</td>
</tr>
<tr>
<td></td>
<td>Adaptation Strategy</td>
<td>activity!adaptation strategy</td>
</tr>
<tr>
<td></td>
<td>Enact Adaptation</td>
<td>activity!enact adaptation</td>
</tr>
<tr>
<td>Artifacts</td>
<td>Requirements Engineering</td>
<td>artifacts!requirements engineering</td>
</tr>
<tr>
<td></td>
<td>Design</td>
<td>artifacts!design</td>
</tr>
<tr>
<td></td>
<td>Construction</td>
<td>artifacts!construction</td>
</tr>
<tr>
<td></td>
<td>Deployment and Provisioning</td>
<td>artifacts!deployment and provisioning</td>
</tr>
<tr>
<td></td>
<td>Operation and Management</td>
<td>artifacts!operation and management</td>
</tr>
<tr>
<td></td>
<td>Adaptation Triggers</td>
<td>artifacts!adaptation triggers</td>
</tr>
<tr>
<td></td>
<td>Adaptation Strategy</td>
<td>artifacts!adaptation strategy</td>
</tr>
<tr>
<td></td>
<td>Enact Adaptation</td>
<td>artifacts!enact adaptation</td>
</tr>
<tr>
<td>Stakeholders</td>
<td>Requirements Engineering</td>
<td>stakeholders!requirements engineering</td>
</tr>
<tr>
<td></td>
<td>Design</td>
<td>stakeholders!design</td>
</tr>
<tr>
<td></td>
<td>Construction</td>
<td>stakeholders!construction</td>
</tr>
<tr>
<td></td>
<td>Deployment and Provisioning</td>
<td>stakeholders!deployment and provisioning</td>
</tr>
<tr>
<td></td>
<td>Operation and Management</td>
<td>stakeholders!operation and management</td>
</tr>
<tr>
<td></td>
<td>Adaptation Triggers</td>
<td>stakeholders!adaptation triggers</td>
</tr>
<tr>
<td></td>
<td>Adaptation Strategy</td>
<td>stakeholders!adaptation strategy</td>
</tr>
<tr>
<td></td>
<td>Enact Adaptation</td>
<td>stakeholders!enact adaptation</td>
</tr>
</tbody>
</table>

http://www.latex-project.org/
to a high quality document, and annotated with the appropriate codes. Finally, this package allows the codes and their corresponding raw data to be exported to a Comma Separated Value (CSV) file. CSV files are easily imported into spreadsheet applications for subsequent analysis.

Once the open coding was complete the codes and their source data were imported into a spreadsheet using the previously mentioned ulQda functionality. Here the codes that were assigned during the open coding process were further refined. Rather than being described by a single code, each passage was broken down into 5 categories: processes, activities, tasks, stakeholders and artifacts. Processes, activities and tasks, in this context, relate the process constructs defined in ISO/IEC 15288 where each process is made up of activities which are in turn made up of tasks. For example, the following passage was assigned the code “activity!operation and management!monitoring” during open coding:

I would monitor the state of the task, the external events, also the state of the information, and the documents and the things that are exchanged.

During the refinement process, a high-level process, an activity and 3 tasks were identified, with no stakeholders or artifacts being identified. The process constructs identified during this example are shown in Table 3.

Once these process attributes were identified they were organized into a conceptually clustered matrix [24]. Similar concepts, in this case the process attributes, were grouped together in to a matrix with activities, stakeholders and artifacts assigned to the appropriate high-level life-cycle processes. They were then ordered into a logical process hierarchy making it easier to identify duplicate or disjointed attributes. Duplicate attributes were translated into one another, while disjointed attributes were reevaluated to see where they might fit into other processes or activities. Finally, the FoR model was updated to reflect the processes, activities, tasks, stakeholders and artifacts identified during the data analysis. As well as additions, V1 of the process model excluded some of the original FoR constructs to reflect the findings of the empirical inquiry. A graphical representation of V1 was constructed using the Business Process Modeling Notation (BPMN) notation, showing the processes and activities. This approach was chosen because there were 38 stakeholders, 35 tasks and 17 artifacts identified which would clutter a complete BPMN

<table>
<thead>
<tr>
<th>Code</th>
<th>Text</th>
<th>Process</th>
<th>Activity</th>
<th>Task</th>
<th>Stakeholder</th>
<th>Artifact</th>
</tr>
</thead>
<tbody>
<tr>
<td>activity!operation and management!monitoring</td>
<td>I would monitor the state of the task, the external events, also the state of the information, and the documents and the things that are exchanged</td>
<td>Operation and Management</td>
<td>Monitoring</td>
<td>Monitor processes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operation and Management</td>
<td>Monitoring</td>
<td>Monitor events</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Operation and Management</td>
<td>Monitoring</td>
<td>Monitor data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Data Analysis Example
representation. Tables were constructed to show the complete set of process attributes for each of the processes in the process model.

### 3.2.4 Evaluation

In order to demonstrate the transferability of the process model, an evaluation was conducted in order to determine if it is compatible with a SBA development life-cycle proposed by Durvasula et al. [26]. Durvasula et al’s life-cycle was collaboratively developed by 10 SBA development practitioners from 8 different companies. During the evaluation, each of the activities from the adaptation process model was mapped to life-cycle processes within Durvasula et al life-cycle. Mapping the activities to the life-cycle without conflict, shows that the process model is suitable for use with an empirically based SBA development life-cycle.

In order to show that the process model is suitable for SBA adaptation it was also compared to a similar process model proposed by Oreizy et al [27][28] for component-based software adaptation. Component based software development has often been referred to as the precursor to service-oriented computing [3], so a process model for adapting component-based applications should share many parallels with a SBA adaptation process model. Oreizy et al’s process model was validated through a worked example where it is used to adapt a cargo routing application.

### 4 FoR Model

In their Systematic Literature Review (SLR), Lane and Richardson [8] identified 8 publications which contain high-level processes and activities for adapting SBAs. Similarly Lane et al [29] identified several additional publications, during an ad-hoc literature search, which contained SBA adaptation activities. Unfortunately, most of the approaches identified the in SLR [8] and the ad-hoc search [29] are non-holistic and lacking the required processes to fully guide SBA adaptation. Each of these publications as well 2 other papers from Bucchiarone et al [30][15], were searched, resulting in a total of 50 SBA adaptation activities (see AppendixA).

Ten generic activities that were identified from the original 50 activities are listed in Table 4. The final step involved in the development of the FoR was to map these generic activities to the S-Cube life-cycle. This results in the FoR model that will be used as a basis for the qualitative research carried in the second part of this study.

The activities and their parent processes were ordered into a sequential workflow which shows how each activity relates to one another. A BPMN diagram is used to visualise the resultant process model. BPMN has a standard modeling notation which
**Generic Activities**

- Adapt SBA
- Define Adaptation Requirements
- Define Adaptation Strategies
- Define Monitoring Requirements
- Deploy Monitoring Mechanisms
- Design Monitors
- Design Adaptation Mechanisms
- Event reasoning from Monitoring
- Implement Monitoring and Adaptation Mechanisms
- Runtime monitoring

### Table 4: Generic Adaptation Activities

---

**Figure 3: FoR Model**

should be understandable by a large number of business as well as technical stakeholders [31]. Figure 3 illustrates the FoR model represented in BPMN in contrast with the block diagram used to construct the S-Cube life-cycle.

A high-level change made was made to the requirements engineering and design process inherited from the S-Cube life-cycle model. This process was separated into two separate processes to accommodate the numerous activities identified for each process.
4.1 FoR Model Description

Our description of the FoR model pays particular attention to the activities and their interrelations. The high level process such as “Requirements Engineering” and “Design” that make up the skeleton of the model are primarily inherited from the S-Cube life cycle model. The S-Cube life cycle is an amended version of Papazoglou and Van Den Heuvel’s [3] service development life cycle with an adaptation cycle added to it’s left hand side [21]. Papazoglou and Van Den Heuvel’s original life-cycle was influenced by approaches such as the Rational Unified Process (RUP) [2], Business Process Modelling (BPM) [32] and Component-Based Development (CBD) [33].

It is important to reiterate at this stage that the FoR model is not a complete life-cycle model as it only contains activities that are related to adaptation. These activities are mapped to the S-Cube life-cycle model which indicates where they should occur during the software development life-cycle. The activities could also be mapped to similar processes within other development life-cycles with similar high level process of the S-Cube life cycle.

4.1.1 Requirements Engineering

In the FoR model we are interested in the capturing adaptation specific requirements for SBAs. Adaptable SBAs will also have many other requirements that are not related to adaptation, but those are outside the scope of this FoR model. Within this FoR there are 2 requirements engineering activities that are specific to adaptation:

**Define Adaptation Requirements** refers to the activity of defining or eliciting requirements relating to why it is necessary for the SBA to adapt. An adaptation requirement may be, for example, if the Quality of Service (QoS) of service A falls below a minimum threshold then the application adapts by choosing an alternative service. The adaptation requirement may also state how the adaptation occurs, adaptation might simply involve choosing between 2 hard coded service end points or in a more complex way by negotiating with a service directory. An adaptation requirement may also specify whether or not the adaptation occurs with user intervention or automatically with minimal or no user input.

**Define Monitoring Requirements** Once adaptation requirements have been determined, monitoring requirements need to be defined in order monitor service attributes that relate to adaptation requirements. For example, in order to enable an application to adapt based on QoS, it is necessary to monitor relevant service quality attributes such as service latency and service reliability. So in this case the
monitoring of service latency and reliability is a monitoring requirement, relating to the adapt if service QoS falls below a minimum threshold adaptation requirement.

4.1.2 Design

There are two adaptation specific design activities within the design process of the FoR:

**Design Monitors** is a design activity where monitors are designed in order to satisfy the monitoring requirements that were defined during the requirements engineering process. Details such as implementation technology and infrastructure need to be considered at this stage. The monitor designs need to be included with a design specification document or a similar alternative that can be used by the application programmers.

**Design Adaptation Mechanisms** Once monitors have been designed, a means to adapt the application based on monitored events must be designed. The adaptation mechanism must have the ability to respond to monitored events and adapt by enacting an adaptation strategy. Bucchiarone et al identified many possible adaptation strategies such as substituting services with known alternatives or service re-negotiation which involves negotiation with a service directory in order to locate alternative services. It is at this point that it must be determined which of these strategies will be used in order to adapt during runtime. Once adaptation strategies have been decided upon the technical and infrastructural details of how they are implemented and triggered need to be designed.

4.1.3 Construction

The adaptation specific construction activity included in the FoR is outlined below:

**Implement Monitoring and Adaptation Mechanisms** This activity involves the implementation or coding of the monitoring and adaptation mechanisms that were discussed in Section 4.1.2. This activity would occur along with all of the other application implementation activities within the software development project. There are no large design features to be decided at this point as they are specified during the design process. However, due to the complex nature of adaptation mechanisms, this implementation in one of the most challenging parts of the development process.

4.1.4 Deployment and Provisioning

The S-Cube life-cycle has inherited its deployment and provisioning process from Papazoglou and Van Den Heuvel’s [3] service development methodology, which itself is influenced by the RUP methodology. The process specifies how a software system is deployed
as well as provisioned or metered, which would allow charging for a service. Here is an outline of the single, adaptation specific, deployment and provisioning activity from the FoR model:

**Deploy Monitoring and Adaptation Mechanisms** The deployment of monitoring and adaptation mechanisms are most likely going to get deployed with the other parts of the SBA and may not require any special treatment. However, if the monitoring or the adaptation mechanisms are built in to a service infrastructure which is external to the application they will have different deployment requirements.

4.1.5 **Operation and Management**

The operation and management of the SBA is one of the most critical processes with respect to adaptation. Whether it be manual or automatic, it is during the operation of the application when certain monitored events prompt the adaptation of the application. Within the FoR model and the S-Cube life-cycle from which it is inherited, the operation and management process is the link between the evolution cycle which undergoes typical maintenance and the adaptation cycle which allows the application to adapt. As well as providing the link between the adaptation and evolution cycles it also has its own activities, and one in particular that relates to adaptation:

**Run-time monitoring execution** Run-time monitoring is a key activity within the operation and management process; it monitors events that signal whether or not run time adaptation should take place. This could be a manual activity or fully automatic depending on the application, but most likely it would be necessary to have some form of involvement by a relevant stakeholder.

4.1.6 **Identify Adaptation Triggers**

Identify adaptation triggers is a process which makes use of monitoring mechanisms in order to identify reasons to adapt. This process occurs during the adaptation cycle, thus all of its sub-activities are related to adaptation. Within this FoR there is one activity in this process which will now be explained:

**Event Reasoning from monitoring execution** is an activity that involves the identification of monitored events from application monitors. These events may be reasoned or identified manually depending the complex of the application. In an fully automated system events would be identified based on application logic, however, in a manual scenario, events might be manually identified by end users or developers through inspection of application logs.
4.1.7 Identify Adaptation Strategies

During this process adaptation strategies are selected to suit the monitoring events that were identified in the previous process. Different event types may be associated with different adaptation strategies, and again, this association may be built in to the applications logic may be made at runtime by a stakeholder. One scenario is that an application has many built in adaptation strategies where the most appropriate one is selected by an end user at runtime.

Select adaptation mechanism to satisfy adaptation need This activity is the only activity within the identify adaptation strategies process. This activity completes all of the goals of its parent process which is to select and adaptation mechanism which is most suited to the monitored event that triggered the adaptation. The specifics actions within this activity are difficult to determine with out having some knowledge about the adaptation engine used in the underlying application.

4.1.8 Enact Adaptation

This process is the last process in the adaptation cycle which simply involves the enactment of the adaptation strategy that was selected. Within the context of this FoR, this process has only one activity which is to adapt the SBA.

Adapt SBA this activity refers to physically adapting the application If it is an autonomous application this activity is automatic. If the application is not autonomous enacting the adaptation of the SBA may require configurations modifications by end users or developers.

5 Model V1

In this section we will present the results of the content analysis that carried out on the 10 interview transcripts (275 minutes of audio) conducted during this study. Altogether there were 198 passages of interview transcript which were assigned codes because they contained information about adaptation processes, activities, tasks, stakeholders, or artifacts. Additionally there were 12 passages of text which were coded because they contained information about the background’s of interview participants.

Once the data analysis was complete the FoR model was updated to represent the study’s findings. Figure 4 is a BPMN representation of the empirically grounded process model. The illustration shows high-level life-cycle processes and activities. Tasks, stakeholders and artifacts are listed separately in the following sub-sections.
5.1 Processes and Activities

The high-level processes from the FoR model used as a starting point for this study remain largely unchanged after the feedback received in the interviews. Most of the interview participants agreed that adaptation activities should occur in the traditional requirements engineering, design, construction, deployment and provisioning and operation and management phases. Additionally several of the participants were of the opinion that there should be a separate testing process or that testing should be incorporated into some of the other processes such as requirements engineering, design and construction. Another notable deviation from the original high-level processes was the lack of emphasis placed on the provisioning sub-process from the deployment and provisioning process.

The purposes of the processes from the adaptation cycle of the FoR model caused confusion amongst some participants during the interviews. There was confusion between terms such as strategies, mechanisms and their applications. These processes as well as their contained activities were modified to make them more intuitive.

Changes to the activities contained within the high-level processes were more extensive
<table>
<thead>
<tr>
<th>Add/Remove</th>
<th>Process</th>
<th>Activity</th>
<th>Data Freq.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remove</td>
<td>Design</td>
<td>Design Adaptation Mechanisms</td>
<td>3</td>
</tr>
<tr>
<td>Add</td>
<td>Design</td>
<td>Design Adaptation Strategies</td>
<td>3</td>
</tr>
<tr>
<td>Remove</td>
<td>Construction</td>
<td>Implement Monitoring and Adaptation Mechanisms</td>
<td>1</td>
</tr>
<tr>
<td>Add</td>
<td>Construction</td>
<td>Implement Adaptation Strategies</td>
<td>1</td>
</tr>
<tr>
<td>Add</td>
<td>Construction</td>
<td>Implement Monitors</td>
<td>1</td>
</tr>
<tr>
<td>Remove</td>
<td>Provisioning</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>Deployment</td>
<td>Deploy Monitoring and Adaptation Mechanisms</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Deployment</td>
<td>Deploy Adaptation Strategies and Monitors</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>Operation and Management</td>
<td>Run-time monitoring execution</td>
<td>3</td>
</tr>
<tr>
<td>Add</td>
<td>Operation and Management</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>Identify Adaptation Triggers</td>
<td>Event Reasoning from monitoring execution</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Identify Adaptation Triggers</td>
<td>Event Reasoning from monitoring</td>
<td>3</td>
</tr>
<tr>
<td>Remove</td>
<td>Identify Adaptation Strategies</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Select Adaptation Strategies</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Remove</td>
<td>Select Adaptation Strategies</td>
<td>Select adaptation mechanisms to satisfy adaptation need</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Select Adaptation Strategies</td>
<td>Select adaptation strategies to satisfy adaptation need</td>
<td></td>
</tr>
<tr>
<td>Remove</td>
<td>Enact Adaptation</td>
<td>Adapt SBA</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Enact Adaptation</td>
<td>Execute selected adaptation strategy</td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Testing</td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

Table 5: V1 Process and Activity Changes

tant than changes the changes to their parent processes. This is to be expected given that the high-level life-cycle processes from the evolution cycle are almost De facto, featuring in prominent life-cycle models such as RUP, ISO/IEC 12207 and the waterfall model [34]. All of the changes made to the processes and activities from the FoR are listed in Table 5. The first column of the table shows whether the processes or activities were added or removed. In most of the cases activities were modified rather than being removed completely. This is shown in the table as a Remove followed by an Add of the modified version. Rows with both a Process and Activity entry refer to the activity and show the process for reference, while rows with just a process entry refer directly to the process. The column Data Freq. contains the number of times data from the interview transcripts support the change in that row. In cases where there is no directly supporting data, changes were made to activities so they correspond to the activities that have supporting data. For example, the modification of “Deploy Monitoring and Adaptation Mechanisms” to “Deploy Adaptation Strategies and Monitors” did not have directly supporting evidence. This activity was changed in order to correspond with the activity “Design Adaptation Strategies” which did have supporting data. The “Provisioning” sub-process was removed from the “Deployment and Provisioning” process, not as a result of comments from the interviewees to remove it, but because when probed, they never offered any details on adaptation activities that need to occur during provisioning.
5.2 Detailed Process Attributes

5.2.1 Requirements Engineering

The 2 Requirements Engineering (RE) activities form the FoR model remain changed in V1, with 3 tasks having being identified for the “Define Adaptation Requirements” activity and 2 for “Define monitoring requirements” (see Table 6). These adaptation activities and tasks are self contained and can be carried out in parallel with the other RE tasks that are necessary for a project. A number of adaptation related artifacts and stakeholders were also identified for the process. Unlike the activities and tasks, the artifacts and stakeholders identified are not exclusively concerned with adaptation. For example, all of the adaptation stakeholders listed here usually have other roles in other RE activities.

<table>
<thead>
<tr>
<th>Adaptation Activities</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define Adaptation Requirements</td>
<td>Define events that trigger adaptation (learning-based, context-based, availability based, rule-based, QoS Based)</td>
</tr>
<tr>
<td></td>
<td>Choose between manual and automatic adaptation</td>
</tr>
<tr>
<td>Define Monitoring Requirements</td>
<td>Validate adaptation requirements</td>
</tr>
<tr>
<td></td>
<td>Define application components that have to be monitored</td>
</tr>
<tr>
<td></td>
<td>Define how monitored events are reported</td>
</tr>
</tbody>
</table>

Table 6: Adaptation Process Attributes for the Requirements Engineering Process

5.2.2 Design

There was one prominent activity change between the FoR and the V1 for the design process. The “Design Adaptation Mechanisms” activity was changed to “Design Adaptation Strategies”. Interviewees felt that Mechanisms was too ambiguous to describe the was in which SBAs adapt:

"Yeah, so I would call these three things out on the right side that I design my monitor, my strategy and my adaptation. I do those three things because my monitor leads to correct strategy".
Adaptation strategies such as service discovery and service substitution were also identified during the inquiry. There were 8 adaptation tasks identified for designing these adaptation strategies, along with 2 tasks for designing monitors (see Table 7). There were 8 adaptation artifacts and 4 adaptation stakeholders also identified for this process. The artifacts and stakeholders identified, like the RE process, are not exclusive to adaptation related activities. In this process artifacts such as “Functional Design” documents are contributed to be adaptation activities, but are also contributed to be non-adaptation activities.

<table>
<thead>
<tr>
<th>Adaptation Activities</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Adaptation Strategies</td>
<td>Design adaptation strategies [service discovery, service substitution]</td>
</tr>
<tr>
<td></td>
<td>Define adaptation workflows</td>
</tr>
<tr>
<td></td>
<td>Design adaptation dashboard (manual adaptation)</td>
</tr>
</tbody>
</table>
|                              | Design logic to match monitored events to adaptation strategies (auto-
|                              | matic adaptation)                                                     |
|                              | Design service interfaces                                            |
|                              | Choose service alternatives                                           |
|                              | Use guide for selecting services                                      |
|                              | Adaptation strategies prototyping                                     |
|                              | Design tests for adaptation strategies                                |
| Design Monitors              | Design monitoring services (Post mortem, real time monitoring)         |
|                              | Determine performance impact of monitors                              |

<table>
<thead>
<tr>
<th>Adaptation Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service topology</td>
</tr>
<tr>
<td>Interface definitions</td>
</tr>
<tr>
<td>Input/Output Data</td>
</tr>
<tr>
<td>Software specifications</td>
</tr>
<tr>
<td>Functional Design</td>
</tr>
<tr>
<td>Enterprise architecture</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adaptation Stakeholders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements engineers</td>
</tr>
<tr>
<td>Solution Architects</td>
</tr>
<tr>
<td>Developers</td>
</tr>
<tr>
<td>Maintainers</td>
</tr>
</tbody>
</table>

Table 7: Adaptation Process Attributes for the Design Process

5.2.3 Construction

The construction activity “Implement monitoring and adaptation mechanisms” was updated to “Implement adaptation strategies” to reflect the new design activity “Design adaptation strategies”. The “Implement Monitors” activity was isolated as a separate activity to show the “Construct monitors” task in its specific context. A noteworthy task identified for the implementation process was “Early Testing” with the other tasks specifically related to construction. The complete adaptation process attribute details for the construction process, including 3 artifacts and 2 stakeholders, can be seen in Table 8.
5.2.4 Testing

The testing process was added to V1 of the model as a result of 3 testing activities that were identified by interviewees. Along with the testing process there were adaptation related testing activities identified for the “Design”, “Construction” and “Deployment” processes. Since the testing activities were identified from scratch, rather than being derived from FoR activities, the interviews did not provide a lot of detail about tasks, artifacts and stakeholders. The activities and single task identified are illustrated in Figure 9.

<table>
<thead>
<tr>
<th>Adaptation Activities</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Functional testing of adaptation configurations</td>
<td>Prove to customer that application will work in all adaptation cases</td>
</tr>
<tr>
<td>System integration testing</td>
<td></td>
</tr>
<tr>
<td>User Acceptance testing of adaptation configurations</td>
<td></td>
</tr>
</tbody>
</table>

Table 9: Adaptation Process Attributes for the Testing Process

5.2.5 Deployment

The “Deployment” process originated as the “Deployment and Provisioning” process from the FoR model. As previously mentioned (section 5.1) the “Provisioning” process was dropped because interviewees did not mention it in relation to adaptation. The process contains one adaptation activity: “Deploy Adaptation Strategies and Monitors” which was changed from “Deploy Monitoring and Adaptation Mechanisms” in order to match the terminology of preceding processes. The deployment of monitors and strategies were
kept as a single activity as the deployment tasks apply generally to both. As illustrated in Table 10, there were 0 adaptation artifacts and 4 adaptation stakeholders identified.

<table>
<thead>
<tr>
<th>Deployment Activities</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deploy Adaptation Strategies and Monitors</td>
<td>Deploy components on separate systems</td>
</tr>
<tr>
<td></td>
<td>Monitor System Dependencies</td>
</tr>
<tr>
<td></td>
<td>Functional Testing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adaptation Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptaion Stakeholders</td>
</tr>
<tr>
<td>Developers</td>
</tr>
<tr>
<td>Maintainers</td>
</tr>
<tr>
<td>Service providers</td>
</tr>
<tr>
<td>Service Consumers</td>
</tr>
</tbody>
</table>

Table 10: Adaptation Process Attributes for the Deployment Process

5.2.6 Operation and Management

The Operation and Management process has retained its monitoring “Run-time monitoring”, from the FoR model, and has gained an activity called “Governance”. The governance activity, as the name implies, contains tasks for governing how the SBA adapts during run-time. The “Run-time monitoring” activity has been updated with tasks for monitoring data, events and processes as well as a user notification task. The “Governance” activity has tasks for the governing the adaptation as well as negotiating Service-Level Agreements (SLAs). A “State Model” was the only adaptation artifact identified without any relevant stakeholders being identified (see Table 11).

<table>
<thead>
<tr>
<th>Operation and Management Activities</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run-time monitoring</td>
<td>Audit service delivery infrastructure</td>
</tr>
<tr>
<td></td>
<td>Notify users of critical events</td>
</tr>
<tr>
<td></td>
<td>Monitor processes</td>
</tr>
<tr>
<td></td>
<td>Monitor events</td>
</tr>
<tr>
<td></td>
<td>Monitor data</td>
</tr>
<tr>
<td>Governance</td>
<td>Govern adaptation</td>
</tr>
<tr>
<td></td>
<td>Negotiate SLAs</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adaptation Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>State Model</td>
</tr>
<tr>
<td>Adaptation Stakeholders</td>
</tr>
</tbody>
</table>

Table 11: Adaptation Process Attributes for the Operation and Management Process

5.2.7 Identify Adaptation Triggers

The “Identify adaptation triggers” process has not changed since V1 of the process model, it still retains the single Activity: “Event Reasoning from monitoring”, with no lower
level tasks. However, this activity was confirmed as a relevant adaptation task during the interviews:

"What are the kind of criteria that we would want for this type of adaptation, the trigger points for doing some adaptation".

As Table 12 illustrates, there were no adaptation artifacts or stakeholders identified during the interview process.

<table>
<thead>
<tr>
<th>Identify Adaptation Triggers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation Activities</td>
</tr>
<tr>
<td>Event Reasoning from monitoring</td>
</tr>
<tr>
<td>Adaptation Artifacts</td>
</tr>
<tr>
<td>Adaptation Stakeholders</td>
</tr>
</tbody>
</table>

Table 12: Process Attributes for the Identify Adaptation Triggers Process

### 5.2.8 Select Adaptation Strategies

The "Select adaptation strategies" process, like the "Identify adaptation triggers" process, retains its original activity from the FoR model. However, in this case the "Select adaptation strategies" activity has been updated with 3 tasks for selecting adaptation strategies. A single stakeholder, "End user", has also been identified for the process (see Table 13).

<table>
<thead>
<tr>
<th>Select Adaptation Strategies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptation Activities</td>
</tr>
<tr>
<td>Select adaptation strategies to satisfy adaptation need</td>
</tr>
<tr>
<td>Select adaptation strategy based on recommendation</td>
</tr>
<tr>
<td>Select adaptation strategy from an end used adaptation dashboard</td>
</tr>
<tr>
<td>Select adaptation strategy to satisfy adaptation need</td>
</tr>
<tr>
<td>Adaptation Artifacts</td>
</tr>
<tr>
<td>Adaptation Stakeholders</td>
</tr>
<tr>
<td>End User</td>
</tr>
</tbody>
</table>

Table 13: Process Attributes for the Select Adaptation Strategies Process

### 5.2.9 Enact adaptation

The final process, "Enact adaptation", has also not had many process attributes identified. The wording of its single activity "Execute selected adaptation strategy" had been reworded to match the terminology of other activities but its intent remains the same. The "End user" stakeholder has also been identified for this process (see Table 14).
Table 14: Process Attributes for the Enact Adaptation Process

<table>
<thead>
<tr>
<th>Adaptation Activities</th>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute selected adaptation strategy</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Adaptation Artifacts</th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Adaptation Stakeholders</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>End User</td>
<td></td>
</tr>
</tbody>
</table>
6 Evaluation

There are two methods employed to determine the external validity of the process model developed during this research. In Section 6.1 the process model will be compared to a previously validated component-based software adaptation process model. This will indicate whether the model contains the required activities for adaptation. In Section 6.2 the activities from the process model will be mapped to an SBA development life-cycle which will demonstrate their transferability.

6.1 Inter-model Evaluation

The purpose of this section is to determine whether or not the model that has been developed in this paper, hereafter referred to as V1, is capable of adapting SBAs. In order to achieve this V1 will be compared to a similar model proposed and validated by Oreizy et al [27][28] for adapting component-based applications. Component-based applications are similar to service-based applications, with both application types being composed of loosely coupled self-contained software modules. This similarity allows comparisons to be made between the two approaches. However, the different development paradigms being addressed by each means that a like for like comparison is not being made.

The two models will be compared at an activity level since Oreizy et al’s model does not specify the life-cycle processes within which its activities occur. Each of Oreizy et al’s activities will be analysed to see if they have equivalent or comparable activities within V1. The degree of similarity between the activities will also be judged by degree of similarity between their component tasks. Oreizy et al’s do not explicitly list the tasks within each activity, however they are described verbally within their paper. These task descriptions have been extracted for the comparative analysis.

6.1.1 High-Level Comparison

Oreizy et al’s adaptation model like V1 has two cycles: an adaptation cycle and an evolution cycle. Both models implement changes to applications during an offline evolution cycle. However, the overall approaches for adaptation are different. In the model that we have proposed, adaptation mechanisms are built into the application at runtime, then changes can be made to the operational application using these mechanisms. Oreizy et al change the composition of the application during evolution, then during adaptation implement the changes enacted during evolution. They argue that making changes during evolution is safer because it allows the application to be tested thoroughly before reflecting those changes in the running system. This is a valid consideration which is addresses in V1 by the task "Prove to customer that application will work in all adapta-
tion case”. This task is completed during the testing process and ensure that possible application configurations are tested off line during evolution before the application is made operational. The differences in the adaptation approach may be rooted in that fact that components are more tightly coupled than services and hence require a greater deal of testing and integrity checks during evolution. The two approaches share many of the same activities, for example, both approaches have runtime monitoring processes to detect when adaptation should be enacted. Both methods also have activities to govern and manage runtime adaptations.

6.1.2 Detailed Comparison

Table 15 was constructed in order to get a better picture of whether V1 contains equivalent adaptation activities and tasks to those outlined by Oreizy et al. The left hand column contains the complete list of activities and tasks from Oreizy et al while the right hand side shows comparable or similar tasks and activities from V1.

It is evident from the table that all of the adaptation activities from Oreizy et al have comparable activities in V1. This shows that, at a high-level at least, V1 contains the basic ingredients for adaptation. The table does not show the adaptation activities identified in V1 which do not have counterparts in Oreizy et al’s model. This shows that V1 goes beyond the basic set of activities for adaptation and considers adaptation more explicitly during evolutionary processes such as design, deployment and testing. There are some activities where Oreizy et al has more detail than V1 particularly with regard to its monitoring activities. The similarities that can be seen between the two models suggest that V1 is a valid adaptation model. This however, is based on the assumption that Oreizy et al’s model is itself valid and that adapting component-based applications is a comparable process to adapting SBAs.

6.2 Transferability

In order to determine the transferability of V1, each of its activities were mapped to a component-based application development life-cycle. This life-cycle proposed by Durvasula et al [26], suggests a high-level process as well as best practice guidelines for developing SBAs. The best practice guidelines are given in a non-sequential format, providing practitioners with a menu of best practices that they can select to use in conjunction with the proposed life-cycle process model. If the activities and tasks from V1 can be intuitively mapped to V1, this shows the transferability of the model to a real-world development scenario.

Table 16 illustrates Durvasula et al’s development life-cycle annotated with the activities from V1 which are highlighted in bold. The granularity of development activities
<table>
<thead>
<tr>
<th>Oreizy et al</th>
<th>Model V1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A1: Maintain consistency and system integrity</strong></td>
<td><strong>Governance/Design Adaptation Strategies/Functional testing of adaptation configurations</strong></td>
</tr>
<tr>
<td>T1: Preserve consistent model of application architecture</td>
<td>Prove to customer that application will work in all adaptation cases</td>
</tr>
<tr>
<td>T2: Preserve strict correspondence between architectural model and execution implementation</td>
<td>Govern adaptation</td>
</tr>
<tr>
<td>T3: Develop architecture evolution manager (AEM) to mediate changes to the AEM</td>
<td>Design adaptation dashboard (manual adaptation)</td>
</tr>
<tr>
<td><strong>A2: Enact Changes</strong></td>
<td><strong>Implement Adaptation Strategies</strong></td>
</tr>
<tr>
<td>T4: Use an interactive editor to construct architectures and describe modifications</td>
<td></td>
</tr>
<tr>
<td>T5: Use design tools to critique architectures as they are constructed</td>
<td>Early Testing</td>
</tr>
<tr>
<td>T6: Use domain specific tools to check for semantic errors</td>
<td>Refinement</td>
</tr>
<tr>
<td>T7: Use modification interpreter to interpret change description language scripts as AEM primitives</td>
<td>Construct adaptation strategies</td>
</tr>
<tr>
<td><strong>A3: Collect observations</strong></td>
<td><strong>Run-time monitoring</strong></td>
</tr>
<tr>
<td>T8: Observe and provide notifications of exceptional events such as resource shortages</td>
<td>Monitor Events/Notify users of critical events</td>
</tr>
<tr>
<td>T9: Dynamic modification of monitors and monitored events</td>
<td></td>
</tr>
<tr>
<td>T10: Model application behavior as patterns of events</td>
<td></td>
</tr>
<tr>
<td>T11: Remote monitoring by human actors</td>
<td>Audit service delivery infrastructure</td>
</tr>
<tr>
<td><strong>A4: Evaluate and monitor observations</strong></td>
<td><strong>Event Reasoning from monitoring/Functional testing of adaptation configurations</strong></td>
</tr>
<tr>
<td>T12: Monitor behaviors of the running application and compare them to behavioral requirements</td>
<td></td>
</tr>
<tr>
<td>T13: Determine all possible architectural configurations to use for consistency checks</td>
<td>Prove to customer that application will work in all adaptation cases</td>
</tr>
<tr>
<td>T14: Automatic runtime consistency checks</td>
<td></td>
</tr>
<tr>
<td><strong>A5: Plan Changes</strong></td>
<td><strong>Define Adaptation Requirements/Define monitoring requirements</strong></td>
</tr>
<tr>
<td>T15: Plan which observations are required for enacting adaptations</td>
<td>Define events that trigger adaptation (learning-based, context-based, availability based, rule-based, QoS based)</td>
</tr>
<tr>
<td>T16: Plan which adaptations to make and when to make them</td>
<td>Define application components that have to be monitored</td>
</tr>
<tr>
<td><strong>A6: Deploy change descriptions</strong></td>
<td><strong>Execute selected adaptation strategy</strong></td>
</tr>
<tr>
<td>T17: Use AEM to translate change descriptions into specific application modifications</td>
<td></td>
</tr>
</tbody>
</table>

Table 15: Oreizy et al vs Model V1

from Durvasula et al’s life-cycle and V1 are somewhat different, with Durvasula et al embedding some life-cycle processes within others, for example, Deployment is embedded as an activity within the IT Operations process. This is not a problem, but it does have some knock on effects to the mappings contained in the Table 16. Some of the activities from V1 are represented as tasks to satisfy the different level of process granularity used by Durvasula et al.

It is evident from Table 16 that the activities from V1 fit seamlessly into the SBA development life-cycle indicating their transferability. Changes in granularity have been made in order to make the mapping. However, this is to be expected given the variation of process attribute granularity in SBA development process models such as those reviewed in [8].
<table>
<thead>
<tr>
<th>Life-Cycle Process</th>
<th>Activity</th>
<th>Tasks</th>
</tr>
</thead>
</table>
| Requirements and Analysis | Map high-level business processes  
Prioritise business services  
Capture business service requirements  
Define Adaptation Requirements  
Define Monitoring Requirements | Architecture Review  
Prioritise and add to solutions portfolio |
| | Review requirements  
Review alternatives and estimate effort  
Propose solution |
| Design and Development | Assign resources to solutions development team  
Design solutions - identify reuse opportunity  
Design Adaptation Strategies  
Design Monitors | Develop, QA and conduct UAT for business solution  
Implement Adaptation Strategies  
Implement Monitors  
Functional testing of adaptation configurations  
System integration testing  
User Acceptance testing of adaptation configurations |
| IT Operations | Assign resources to service operations teams  
Identify infrastructure needs and establish systems environment  
Deploy business solution  
Maintain solution to business requirements | Deploy Adaptation Strategies and Monitors  
Run-time monitoring  
Governance |
| | Identify Adaptation Triggers  
Event Reasoning from monitoring |
| | Select Adaptation Strategies  
Select adaptation strategies to satisfy adaptation need |
| | Enact Adaptation  
Execute selected adaptation strategy |

Table 16: Mapping V1 to Durvasula et al’s SBA Life-Cycle

7 Discussion

The research presented in this paper has addressed a gap. In the first instance, while there have been many implementation techniques proposed for adapting services and SBAs, they are not process focused. Secondly, studies have not been based on the experiences and opinions of expert SBA development practitioners. When developing a process model for adapting SBAs it is important to look at the broader perspective of the entire SBA development life-cycle, taking into account the various actors and stakeholders involved. By analysing interviews carried out with developers, analysts, architects, and business stakeholders from various domains, a useful real-world SBA adaptation process model has been constructed.

The approach taken in this paper has leveraged the existing literature, which was used to construct a conceptual framework to guide the fieldwork and qualitative data analysis. There are arguments against this approach, with some suggesting [35] that the use of a conceptual framework influences research participants and results in biased findings. Miles and Huberman [24], however, argue that all qualitative research is inherently biased by the researcher and that the benefits of using a conceptual framework outweigh the drawbacks. In completing this study, we found that participants offered insights that they might not
have thought about in the absence of prompts from the conceptual framework.

The data analysis conducted followed a best practice positivistic approach which allows the results to be audited and reviewed by secondary researchers. This rigorous, repeatable approach aims to eliminate researcher bias and enhance the credibility of the research. Furthermore, this approach allows for the findings to be extended by conducting additional interviews and analysing the additional data with the original analysis method.

It was discovered during the interview process that participants were able to readily relate to the processes in the evolution cycle and to provide good insights into the adaptation activities that should occur during these phases. This is likely to be due to the fact that these life-cycle processes are present in many of the formal development life-cycles that are reported in the literature. Respondents found it more difficult to relate to the processes introduced in the adaptation cycle, which resulted in significantly less data being collected for these processes. Another interesting finding was that many practitioners did not consider the use of a service specific life-cycle. They normally develop SBAs using traditional waterfall style methods. However, on reviewing the frame of reference model with an additional adaptation cycle they considered this to be a useful approach.

### 7.1 Threats to Validity

The primary threat to the validity of this research in the authors’ opinion is the generalisability of the adaptation model constructed. Every effort was made to select interview partners from a diverse set of companies and a diverse set of roles, but this does not guarantee that the results are generally applicable. In order to mitigate these risks the process mode was mapped to a SBA development life-cycle that was empirically developed independently of this research. The goodness of fit that was achieved during the mapping helps to show the model can be generally applied.

In the absence of having used the model in a real-life scenario it is difficult to determine its capability of adapting SBAs. There are two conceivable methods of determining its real world adaptation capabilities, the first being to actually use it in a real adaptable SBA development project, the second to compare it to a similar model that has been validated in a real scenario. The latter option was chosen mainly for practical reasons. While it is arguably better to validate the model in real scenario, there is also a lot to be learned by comparison with an existing approach. In this paper the comparison shows that the adaptation process model does contain the activities required to safely adapt an SBA during runtime.

The final threat considered is whether or not the data collected reached a point of saturation where activities and tasks identified begun to re-occur in the later interviews.
The later interviews did repeat much of the earlier findings with the exception of the life-cycle processes in the adaptation cycle. Had the interview questions relating to the adaptation cycle been focused on more during the interviews the would have resulted in more data. The problem seemed to be that the questions for the evolution cycle often took a lot of time and by the time the adaptation cycle questions came around the respondents enthusiasm diminished resulting in less data being collected. A solution may have been to focus solely on the adaptation cycle in later interviews, however, this would have been at the cost of valuable data collected by the evolution related questions. We plan to take this approach in future research.

7.1.1 Validation

Lincoln and Guba [36] argue that qualitative research should be internally valid, externally valid, reliable and objective. In this paper we focused on internal and external validity. Internal validity relates to “how” the research is carried out, and whether the methods used are credible. Two suggested methods of ensuring internal validity are data source and method triangulation [37]. Externally validity refers to the transferability of the research results to other similar contexts. If research results are transferable then they are arguably much more useful than results that are specific to a particular context.

The internal validity of this research has been sought through the use of data source and research method triangulation. There were two data sources used to construct the process model, the FoR was constructed with data from relevant literature sources, while the empirically grounded version of the model was constructed using data gathered during field work. There were also two research methods employed, survey research where individuals were interviewed and case study research where the development process of SbaSoft was documented through interviews with several practitioners working at SbaSoft.

7.2 Conclusions and Future Work

During this research, the authors conducted a review of the literature and identified adaptation activities that could be used to adapt SBAs. These activities in combination with a skeleton life-cycle model proposed by the S-Cube consortium [21] formed the basis for a frame of reference process model for adapting SBAs. This frame of reference was used to guide interviews with SBA development practitioners who had experience with or who could provide expert opinion on how to adapt SBAs. The data that was collected in these interviews was transcribed and analysed using qualitative content analysis techniques. The resulting adaptation activities and tasks were constructed into a detailed process model identifying the relevant stakeholders and development artifacts for each stage of the process. The model’s transferability and capability were demonstrated during an
evaluation process where the model was systematically compared to a component-based application adaptation model and an empirically based SBA development life-cycle. The approach taken in this paper has advantages over similar approaches in that its process focused and is based on input provided by experts from the field.

Future work will focus in refining the process model and investigating whether or not the activities identified for the adaptation cycle can be enriched with more detail. In order to determine the applicability of the model in the field, efforts will be made to use the model during an adaptable SBA development project.
## A Adaptation Activities

<table>
<thead>
<tr>
<th>Source</th>
<th>Adaptation Activity</th>
<th>Generic Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lane et al</td>
<td>Monitor message sequences amongst services and its partners</td>
<td>Runtime monitoring</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Runtime service discovery</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Requirements and analysis stage: define KPIs and management policies</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Service specification: identify the service properties to specify</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Specifying service decision model</td>
<td>Define Adaptation Strategies</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Set warning thresholds and alerts for compliance failures</td>
<td>Define Monitoring Requirements</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Specify monitoring rules according to the adopted SeCSE monitoring language (SECMOL)</td>
<td>Define Monitoring Requirements</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Service deployment: deploy the monitoring rules and recovery policies within the system</td>
<td>Deploy Monitoring Mechanisms</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Gather QoS metrics on the basis of SLAs</td>
<td>Design Monitors</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Evaluate SLA QoS metrics</td>
<td>Design Monitors from Monitoring</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Readjust service weights for request queues</td>
<td>Design Monitors</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Designing service adapters</td>
<td>Dosing Adaptation Mechanisms</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Recovery management: identify, by looking at the monitoring data, the need for a recovery action</td>
<td>Event reasoning from Monitoring</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Detect protocol violations</td>
<td>Runtime monitoring</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Monitor service, application, middleware, OS, hardware, and network</td>
<td>Runtime monitoring</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Monitor workloads</td>
<td>Runtime monitoring</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Insert monitoring rules and recovery actions in concrete parts</td>
<td>Runtime monitoring</td>
</tr>
<tr>
<td>Lane et al</td>
<td>Monitor services</td>
<td>Runtime monitoring</td>
</tr>
<tr>
<td>Bucchiarone et al</td>
<td>Define adaptation and monitoring requirements</td>
<td>Define Adaptation Requirements/Define</td>
</tr>
<tr>
<td>Bucchiarone et al</td>
<td>Design for monitoring and adaptation</td>
<td>Monitoring Requirements/Define</td>
</tr>
<tr>
<td>Bucchiarone et al</td>
<td>Construction of monitors and adaptation mechanisms</td>
<td>Implement Monitoring and Adaptation</td>
</tr>
<tr>
<td>Bucchiarone et al</td>
<td>Deployment-time adaptation</td>
<td>Design Monitors</td>
</tr>
<tr>
<td>Bucchiarone et al</td>
<td>Run-time Monitoring</td>
<td>Design Monitors from Monitoring</td>
</tr>
<tr>
<td>Bucchiarone et al</td>
<td>Decide between adaptation and evolution</td>
<td>Design Monitors</td>
</tr>
<tr>
<td>ProDAOSS</td>
<td>Organisational services realization paths are documented by a dynamic service hypergraph</td>
<td>Design Adaption Requirements</td>
</tr>
<tr>
<td>ProDAOSS</td>
<td>Organisational services are designed as service centers in the architectural design discipline</td>
<td>Design Adaption Requirements</td>
</tr>
<tr>
<td>ProDAOSS</td>
<td>Services realization environment is open and adaptable through the use of a reinforcement learning algorithm and a probabilistic reputation model</td>
<td>Event reasoning from Monitoring</td>
</tr>
<tr>
<td>PLASTIC</td>
<td>Runtime Analyser</td>
<td>Runtime monitoring</td>
</tr>
<tr>
<td>PLASTIC</td>
<td>SLA monitor</td>
<td>Runtime monitoring</td>
</tr>
<tr>
<td>PLASTIC</td>
<td>On-line validation</td>
<td>Runtime monitoring</td>
</tr>
<tr>
<td>BICDF</td>
<td>Take enterprises business and technical requirements as well as dependencies between them into consideration</td>
<td>Define Adaption Requirements</td>
</tr>
<tr>
<td>BICDF</td>
<td>Enterprieses describe their purpose and high level requirements for a business collaboration</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>BICDF</td>
<td>Define the operational conditions under which businesses can cooperate</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>BICDF</td>
<td>Negotiation agreement describing the interactions among the services from the different parties</td>
<td>Dosing Adaptation Mechanisms</td>
</tr>
<tr>
<td>Chang</td>
<td>Defining Target Services</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>Chang</td>
<td>Defining Unit Services</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>Chang</td>
<td>Planning service component acquisition</td>
<td>Dosing Adaptation Mechanisms</td>
</tr>
<tr>
<td>Chang</td>
<td>Acquiring service components</td>
<td>Dosing Adaptation Mechanisms</td>
</tr>
<tr>
<td>Chang</td>
<td>Composing services</td>
<td>Adapt SBA</td>
</tr>
<tr>
<td>Multi-view SOAD</td>
<td>Develop use case models</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>Multi-view SOAD</td>
<td>Service identification by viewpoint</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>Multi-view SOAD</td>
<td>Service interface conception</td>
<td>Dosing Adaptation Mechanisms</td>
</tr>
<tr>
<td>Multi-view SOAD</td>
<td>Mapping to platform specific models and code generator</td>
<td>Event reasoning from Monitoring</td>
</tr>
<tr>
<td>CSOMA</td>
<td>Model services with variability points</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>CSOMA</td>
<td>Select orchestration schema based on the context of the incoming request</td>
<td>Event reasoning from Monitoring</td>
</tr>
<tr>
<td>CSOA</td>
<td>Design adaptation views for Platform Specific Models (PSMs)</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>Dino</td>
<td>Develop UML2 Models Model</td>
<td>Define Adaptation Requirements</td>
</tr>
<tr>
<td>Dino</td>
<td>Generate requirements and capabilities of broker services</td>
<td>Dosing Adaptation Mechanisms</td>
</tr>
</tbody>
</table>

Table 17: Adaptation Activities
B  Interview Guide
Instructions
Before completing this interview make sure respondent has read the background document which explains some of the concepts as well as terminology used.

Each of the questions in the survey will relate to the life-cycle shown in Figure 1 which will be used as a frame of reference for the questions.

![Figure 1 Process for adapting Service-Based Applications (Frame of Reference)](image)

Questions

Section 1 Working Environment

1. What is your current position?
2. Do you work with adaptable Service-Based Applications (SBAs)?
3. Do you think a coherent life-cycle for adaptable SBAs would be beneficial?

Section 2 Detailed Domain Knowledge

4. At what stage in the SBA development life-cycle should one define Adaptation requirements?
5. At what stage in the SBA development life-cycle should one define Monitoring requirements?
   a) Stakeholders?
   b) Activities?
   c) Artefacts?

6. At what stage in the SBA development life-cycle should one design monitors?
   a) Stakeholders?
   b) Activities?
   c) Artefacts?

7. At what stage in the SBA development life-cycle should one design adaptation mechanisms?
   a) Stakeholders?
   b) Activities?
   c) Artefacts?

8. At what stage in the SBA development life-cycle should one implement monitoring and adaptation mechanisms?
   a) Stakeholders?
   b) Activities?
   c) Artefacts?

9. At what stage in the SBA development life-cycle should runtime monitoring take place?
   a) Stakeholders?
   b) Activities?
   c) Artefacts?

10. At what stage in the SBA development life-cycle should event reasoning from monitoring execution take place?
   a) Stakeholders?
   b) Activities?
   c) Artefacts?

11. At what stage in the SBA development life-cycle should adaptation mechanisms be selected to satisfy adaptation needs?
   a) Stakeholders?
   b) Activities?
   c) Artefacts?

12. At what stage in the SBA development life-cycle should adaptation be enacted?
References


On The Evolution of Services
Vasilios Andrikopoulos, Salima Benbernou, and Michael P. Papazoglou

Abstract—In an environment of constant change and variation driven by competition and innovation, a software service can rarely remain stable. Being able to manage and control the evolution of services is therefore an important goal for the Service-Oriented paradigm. This work extends existing and widely-adopted theories from software engineering, programming languages, service oriented computing and other related fields to provide the fundamental ingredients required to guarantee that spurious results and inconsistencies that may occur due to uncontrolled service changes are avoided. The paper provides a unifying theoretical framework for controlling the evolution of services that deals with structural, behavioral and QoS level-induced service changes in a type-safe manner, ensuring correct versioning transitions so that previous clients can use a versioned service in a consistent manner.

Index Terms—Services Engineering, Service Evolution, Versioning, Service Compatibility

I. INTRODUCTION

SERVICES are subject to constant change and variation. Service changes originate from the introduction of new functionality to a service, the modification of already existing functionality to improve performance or the inclusion of new policy constraints that require that the behavior of services be altered. Such changes lead to a continuous service redesign and improvement effort. However, they should not be disruptive by requiring radical modifications in the very fabric of existing services and applications.

Changes can happen at any stage in the service life cycle and have an unpredictable impact on the service stakeholders [1]. Being therefore able to control how changes manifest in the service life cycle is essential for both service providers and service consumers.

Service evolution is the disciplined approach for managing service changes and is defined as the continuous process of development of a service through a series of consistent and unambiguous changes [2]. Service evolution is expressed through the creation, provisioning and decommissioning of different variants of the service – called versions – during its life time. These versions must be aligned with each other in such a way as to allow a service developer to track the various modifications introduced over time and their effects on the original service. To control service development, a developer needs to know why a change was made, what its implications are, and whether the resulting service version is consistent and does not render its consumers inoperable.

Eliminating spurious results and inconsistencies that may occur due to uncontrolled changes is thus a necessary condition for services to evolve gracefully, ensure service stability, and handle variability in their behavior. With the above backdrop, we can classify service changes on the basis of their causal effects as:

1) Shallow changes: Small-scale, incremental changes that are localized to a service and/or are restricted to the consumers of that service.
2) Deep changes: Large-scale, transformational changes cascading beyond the consumers of a service possibly to consumers of an entire end-to-end service chain.

Deep changes require an approach dealing with shallow changes, which form their foundation, as well as with intricacies of their own. For this purpose deep changes rely on the assistance of a change-oriented service life cycle methodology to respond appropriately to changes [2]. They are predominantly concerned with analyzing the effects and dealing with the ramifications of operational efficiency changes and changing compliance requirements which rely on service composition reengineering exercises. Deep changes therefore constitute a challenging and open research problem in the context of service engineering. In this work we focus on developing a sound theoretical framework and approach for dealing with shallow changes as a precursor to dealing with deep changes.

Ensuring that a change is shallow requires service developers to reason about the effect of change both on service providers as well as service consumers. The number, type and specific needs of the consumers is often unknown and their dependencies on the service are transparent to the developer. This reasoning can only be performed on the basis of formal principles and theories that control shallow changes. Such an approach for controlling shallow changes is currently unavailable and this work aims to address this need. Without a comprehensive formal framework for controlling and delimiting service evolution, service versioning cannot succeed. The goal of this research is therefore to provide a theoretical framework to assist service developers in their effort to develop evolving services while constraining the effects of changes so that they do not spread beyond an evolving service. In this way, service changes are kept localized, so that they neither lead to inconsistent results, nor do they disrupt service clients. In particular, we shall constrain our work to addressing the effects of shallow changes on service interface specifications.

Services typically evolve by accommodating a multitude of changes along the following, non-mutually exclusive dimensions:

1) Structural changes focus on changes that occur on the service data types, messages and operations, collectively known as service signatures.
2) Behavioral changes affect the business protocol of a service. Business protocols specify the external messaging and perceived behavior of services (viz. the rules that
govern the service interaction between service providers and consumers) and, in particular, the conversations that the services can participate in.

3) **Policy-induced changes** describe changes in policy assertions and constraints on the invocation of the service. For instance they express changes to the Quality of Service (QoS) characteristics and compliance requirements of a service.

Structural, behavioral and QoS-related policy-induced changes refer to the externally observable aspects of a service (in terms of its signatures, protocols, etc.). These types of changes have a direct and profound impact on the service interfaces and as such they will be discussed extensively in the following sections. Changes due to legislative, regulatory or operational requirements on the other hand are typically deep changes [2], and therefore outside the scope of this work.

The contribution of this paper is twofold:
- a language-independent, theoretically-backed approach which brings together structural, behavioral and QoS-related service changes, and
- a rigorous formal framework, type safety criteria and algorithms which control and delimit the evolution of services. The goal of this framework is to assist service developers in controlling and managing service changes in a uniform and consistent manner.

For this purpose we develop a set of theories and models that unify different aspects of services (description, versioning and compatibility). In particular, we provide a consistency theory for guaranteeing type-safety and correct versioning transitions so that previous clients can use a versioned service in a consistent manner.

The theoretical framework presented in this paper is at the cross-section of programming language theories, service oriented computing and software engineering. It provides an innovative approach that challenges and redefines the state of the art in service evolution. At the same time it calls into question the existing standards and support technologies as regards the facilities they provide for service change.

The rest of this paper is organized as follows: Section II discusses the background of our work by presenting a classification of existing service versioning approaches and their techniques for compatible service evolution. Section III presents a running example based on an industrial case study used throughout the rest of the paper. Section IV formally defines the compatibility of services and presents our theoretical framework for the compatible evolution of services. Section V demonstrates the application of the framework to the empirical guidelines supporting the existing approaches and to the evolution scenarios of Section III. In Section VI we summarize our implementation effort. A qualitative evaluation and empirical validation of the framework is performed in Section VII. Finally, Sections VIII and IX discuss related work, and conclusions and future work respectively.

II. BACKGROUND

By its definition, service evolution has two important facets: recording the continuous process of service development, and controlling the consistency and unambiguity of its different versions. The following sections examine these two facets.

A. Service Versioning

Versioning as a concept has its roots in the **Software Configuration Management (SCM)** field that, together with type theory, has contributed in major ways to software maintenance and evolution [3]. From the aspects developed under SCM, of particular interest for service evolution is development support in terms of versioning as summarized in [4]. More specifically, versioning refers to the keeping of a historical record of software artifacts as they undergo change. The reliance of Service Oriented Architecture (SOA) on the publishing of service interface descriptions (e.g. in WSDL 2.0) and interaction protocols (in Abstract BPEL), together with the predominant use of XML as the description language, adds an additional dimension to the versioning of services.

Versioning support during service development has two dimensions:

- **Interface versioning**: versioning support for the service description, i.e. the artifacts that describe the interaction of the service with its environment (e.g. definitions of data types in XML Schema and WSDL and Abstract BPEL documents).
- **Implementation versioning**: versioning support for the code, resources, configuration files and documentation of a service.

Implementation versioning is by definition an SCM issue and as such the techniques from this domain can be applied to it (as discussed in [4]). Traditional SCM systems, such as the popular revision control systems CVS\(^3\) and Subversion\(^4\), or their modern distributed counterparts like GIT\(^5\) and Bazaar\(^6\) (among others) can be used for this purpose. Service implementation is outside the scope of this work and as such it is not considered in the rest of this paper. In the following we summarize the proposals of existing service interface versioning approaches.

1) **Service Version Naming**: Versioning approaches typically distinguish between (consumer) breaking and non-breaking changes (cf. [5] and [6]). The former constitute major releases and the latter minor ones. Naming a version usually follows the **Major#.Minor#** scheme where the sequential major release version number precedes the minor one; version “1.3” for example denotes the 3rd minor revision of the 1st major release (with “1.0” signifying the first version of that release). An alternative naming scheme incorporates a release date stamp instead of the sequence identifier [7].

Such naming schemes do not provide information about the position of the versions in the version graph which represents

---

\(^1\) Web Services Description Language (WSDL) Version 2.0 http://www.w3.org/TR/wsdl20

\(^2\) Web Services Business Process Execution Language (BPEL) http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

\(^3\) http://www.nongnu.org/cvs/

\(^4\) http://subversion.apache.org/

\(^5\) http://git-scm.com/

\(^6\) http://bazaar-vcs.org/
the relationships between versions [8]. In the VRESCo approach [9], the version graph is explicitly stored in the service registry, while in the WSDL and UDDI extension approach of [10], the versioning graph can be reconstructed using the custom meta-data of the annotated service description files.

2) Service Versioning Methods: Versioning in Web services is supported through the mechanisms offered by XML and XML Schema. More specifically, we can distinguish between the following service versioning methods:

1) New XML namespaces for each (major) version.
2) Version Identifiers (VIDs) unambiguously naming a version.
3) A combination of the above.

Approaches like [5], [10] that follow the new XML namespace technique purposely break the consumers of the service by assigning a different namespace to either the service itself or to its data types. The new namespace results in disrupting the binding of the service on the consumer side. New namespaces are therefore meant to be used only if a major version of a service is deployed. On the other hand, VIDs are used either as attributes (either in the root element of the document or in each element separately) [7], [6] or as part of the (endpoint) URL [11], [12]. This, however, requires the consumers to be somehow able to process the versioning information and understand the implications of the naming scheme for their application. Both types of approaches rely on the Major#.Minor# naming scheme either directly as a VID or by incorporating it into the namespace itself. They are not mutually exclusive and they can be used in conjunction for versioning control.

Some of the approaches to service interface versioning use a service registry mechanism like the Universal Description Discovery and Integration (UDDI) standard [13] or a custom registry [9] for storing and controlling the versioning information – either as an alternative, or complementary to the XML-based techniques discussed above. For this purpose they propose the addition of versioning metadata in the service description model that the registry is using.

3) Service Versioning Strategies: The versioning strategy varies depending on the goal of each approach with respect to the breaking or not of consumers. A number of approaches do not consider whether changes to a service version break the consumers of the service, e.g. [7] and [10]. As such, they leave to the developers the prerogative and responsibility of checking whether changes break the service consumers, but they also maintain a high degree of flexibility in the cases they can handle. In principle, these approaches allow for multiple versions of a single service to be accessible at a time.

The majority of existing approaches (e.g. [5], [12], [13]) propose a common compatibility-oriented strategy for versioning: maintain multiple active service versions for major releases, but cut maintenance costs by grouping all minor releases under the latest one. Nevertheless, the cost of maintenance varies in proportion to the number of active versions at a time. The creation of a major version therefore, apart from possibly breaking existing consumers, also increases the effort required for managing the service portfolio.

For this reason, approaches like [6] and [9] take special interest in discussing different decommissioning strategies for non-current versions of the service. Despite differences in the details, the goal in each case is to decrease the number of active versions to the absolute minimum required to serve the service consumers. Usually, a grace period is given before decommissioning a service version and, depending on the change identification model used (see below), either the clients are notified in advance or they have to “discover” for themselves this information.

4) Change Identification Model: In a similar fashion to versioning strategies, the model of how service changes are perceived and identified by the service consumers varies according to the goals of the approach. This model can be classified in one (or more in the case of [14] and [12]) of the following categories: client, notification and transparent.

In the client model [5], [6], [10], both non-breaking and breaking changes result in a new version, and the identification of existence of this version is left to the consumer. The consumer is then required to adapt to the new version if necessary. In the notification model [13], the consumer is explicitly notified for the existence of a new version and asked to take action, usually within a given time period. This typically requires service consumers to subscribe to some sort of notification service. Finally, approaches that enforce non-breaking changes do not have to inform their consumers of changes since in theory the changes are transparent to them. In reality however, some of these approaches allow their clients to identify a new version using one or a variation of the methods above [15], [9].

5) Discussion: The investigation into existing service versioning approaches shows that the ease of use of VIDs in XML, in conjunction with the XML namespace disambiguation mechanism, is more than sufficient for recording and communicating the different versions of the service to its clients. In this respect, it is not necessary to provide additional methodologies for managing the versioning of services.

Since the goal of this work is to provide a service development approach that does not break the existing consumers, the emphasis is on enforcing the transparency of evolution. Major versions should be kept to a minimum and minor versions should be bundled together under one major version in order to minimize the cost of maintenance and achieve a reasonable tradeoff between flexibility and constrained evolution. For this purpose we provide a theoretical framework that allows for sound service evolution. The following sections assume that a robust versioning mechanism is already in place on the basis of which we explain how to analyze and evaluate the changes that lead to different versions of a service description.

B. Compatibility of Service Versions

Compatibility is a concept closely related to versioning. It is usually decomposed into backward and forward cases. A definition of forward and backward compatibility with respect to languages and message exchanges between producers and consumers is given in [16]. Forward compatibility means a new version of a message producer can be deployed without
on the semantics of the message ignored during the processing of a message, without any effect (decentralized) additions to an existing service. The underlying changes to the service, extensions are by definition compatible versions can be either compatible or incompatible (centralized)

relevant notion to both versioning and compatibility: whereas that is not defined in the current version of the language; extensibility provides mappings from documents in any extended set to documents already defined [16]. Extensibility therefore is a relevant notion to both versioning and compatibility: whereas versions can be either compatible or incompatible (centralized) changes to the service, extensions are by definition compatible (decentralized) additions to an existing service. The underlying assumption in all cases is that the additional data can be safely ignored during the processing of a message, without any effect on the semantics of the message

1) Forward Compatibility: Some of the approaches discussed in the previous section (in particular, [6], [15], and [14]) enforce forward compatibility through the use of extensibility. Extensibility is the property of a language to allow information that is not defined in the current version of the language; extensibility provides mappings from documents in any extended set to documents already defined [16]. Extensibility therefore is a relevant notion to both versioning and compatibility: whereas versions can be either compatible or incompatible (centralized) changes to the service, extensions are by definition compatible (decentralized) additions to an existing service. The underlying assumption in all cases is that the additional data can be safely ignored during the processing of a message, without any effect on the semantics of the message [16].

2) Backward Compatibility: Backward compatibility is in practice a mechanism for distinguishing between major and minor releases – as long as the changes applied to a service lead to backward compatible versions of the service they can be considered minor releases, otherwise they are major. The usual approach for defining what constitutes a backward compatible change to a service has been to enumerate the possible compatible changes. This results to a list of permissible and prohibited changes usually, but not exclusively, to the WSDL document describing the service. This list reflects a combination of common sense, technological limitations and empirical findings that results into a set of best practices – guidelines to be followed and not necessarily undisputed rules. These guidelines are presented in Table I and aggregate the guidelines from [5], [11] and [14].

All the changes are expressed in terms of changes to WSDL and XML Schema elements. In summary, the backward compatible changes are only additions of optional elements (either input data types or operations) or the modification of service implementation (as long as it does not affect the WSDL document). The removal or any kind of modification to an operation element is strictly prohibited, as is the modification of the message data types (with the exception of addition of optional data types).

This guideline-based approach is easily applicable and requires minimum support infrastructure and for this reason it is widely accepted. However it exhibits certain disadvantages, the main of which is that it depends on service developers for deducing what is compatible and acting accordingly. Even if these rules are codified and embedded into a service development/versioning tool, as for example in the case of [17], they will always be limited by two factors: their dependency on technology (WSDL in this case) and their lack of a solid theoretical foundation. In our approach we extend the reasoning behind the backward compatible guidelines and we enhance it by showing how these rules can be generated as the result of a theoretical framework to control and delimit service evolution.

### III. Running Example

In order to demonstrate the practical applicability of our work we chose to use the industrial-strength Automotive Purchase Order Processing use case. The use case was developed in conjunction with IBM Almaden and is used as one of the validation scenarios in the S-Cube Network of Excellence [18]. The scenario is based on the cross-industry, standardized Supply Chain Operations Reference (SCOR) model that provides abstract guidelines for building supply chains. The SCOR model comprises of four levels of processes (scope, configurations, business activities and implementation, respectively).

This Automotive Purchase Order Processing use case is an example of how to realize SCOR level 3 activities using SOA-based processes for an enterprise in the automobile industry called Automobile Incorporation (a.k.a. AutoInc). AutoInc consists of different business units, e.g. Sales, Logistics, Manufacturing, etc., and collaborates with external partners like suppliers, banks and transport carriers. The use case describes a typical automobile ordering process, where customers can place automated orders with AutoInc. For a complete description of the use case the reader is referred to [18]. In the following, we present one of the services that is part of the use case and discuss two possible evolution scenarios for it.

#### A. Purchase Order Processing Service

The Purchase Order Processing service (POSERVICE) supports the “receive purchase order” activity of the Sales unit of AutoInc. POSERVICE is at the core of the use case and has a critical function. In case of failure or underperformance, the whole chain of interlinked services in the use case will be detrimentally affected.

---

**TABLE I**

GUIDELINES FOR BACKWARD COMPATIBLE CHANGES

<table>
<thead>
<tr>
<th>Change</th>
<th>Backwards Compatible</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add (Optional) Message Data Types</td>
<td>Yes (input only)</td>
</tr>
<tr>
<td>Add (New) Operation</td>
<td>Yes</td>
</tr>
</tbody>
</table>
| Modify Service Implementa
tion\(^a\) | Yes                  |
| Remove Operation        | No                   |
| Modify Operation\(^b\)  | No                   |
| Modify Message Data Types | No                |

\(^a\)As long as it has no effect on the service interfaces.

\(^b\)Includes renaming, changing parameters, parameter order and message exchange pattern.

---


Non-functional properties are:

- **Performance**: The ability of a service to perform its required functions under stated conditions for a specified period of time. It is the overall measure of a service to maintain its service quality.

In a hypothetical case, we assume that latency is expected to vary between 0.15 and 0.3 seconds, availability to vary between 80 and 95% and performance to be at minimum 90% for the same conditions.

### B. Evolution Scenarios

Since the goal of this work is to study the evolution of services, it is only natural to perceive PO SERVICE itself as subject of change. In order to illustrate possible evolutionary paths that the service can take during its life time, we describe two evolution scenarios: a relatively simple service improvement scenario and a more complicated service redesign scenario.

#### a) Service Improvement Scenario

For this scenario we assume that the service developers attempt to improve the QoS characteristics of the service by aiming for lower latency and focusing on less error-prone handling of incoming requests. For this purpose a new version of the PO SERVICE is designed, where:

- The customers, both new and returning, have to provide the delivery information along with the purchase order. In this way, the disruption caused by missing information in later stages of the process is minimized.
- In order to streamline and accelerate the servicing time of each order, the service forwards the delivery information to an intermediate service in the Logistics unit of AutoInc, in order to verify that the address does not contain an error, and that it points to an existing delivery address.

The new version asks customers to always include in their requests the DeliveryInfo element, resulting in changing its multiplicity as shown in Fig. 3. The rest of the WSDL file remains as is. Furthermore, we assume the changes in the scenario result in changing the non-functional characteristics of the PO SERVICE by introducing for example lower latency.

Fig. 1 contains the WSDL definition of the service. Starting from its port types, PO SERVICE is communicating with its consumers in an asynchronous manner through the receivePO and receivePOCallBack operations. The actual protocol for communicating with the service is defined in BPEL. This is shown in Fig. 2, where a two-step interaction between PO SERVICE and its consumers is explicitly defined. The consumer is supposed to invoke the receivePO operation with the Purchase Order document, codified by the FODocument data type, and wait for the call back invocation receivePOCallBack from the service side with the acknowledgement of the order receipt. The simple message payload for the operations of the service facilitates the demonstration of the theoretical constructs we develop.

Finally, for the non-functional aspects of PO SERVICE, and given the absence of a widely acceptable standard for the characterization of non-functional properties, we use the the SCube Quality Reference Model (QRM) [19]. In particular, the QRM characteristics used for the definition of the PO SERVICE non-functional properties are:

- **Availability**: The degree of availability of the service to its consumers relative to a maximum availability of 24 hours, seven days a week.
- **Latency**: Time passed from the arrival of the service request until the end of its execution/service.

Fig. 2. PO SERVICE definition in BPEL.
ranging from 0.075 secs to 0.15 secs, but worse performance of a minimum of 81% (due to additional service communication overheads).

b) Service Redesign Scenario: In this scenario, we assume a potential customer of POSERVICE requests to use a synchronous communication pattern with the service for application safety reasons. The service developers take the opportunity to also address some standing issues of the service and redesign it in two ways:

- In order to accommodate both existing consumers and the new customer, the service provides both synchronous and asynchronous interfaces. However, instead of running two versions of the service in parallel, both types of interaction with the service are grouped into one communication protocol. The new protocol allows consumers to decide which type of communication to use at its entry point. This allows for the synchronous interface to use the same message types as the asynchronous one since they carry the same payload.
- The time stamp on incoming messages is not necessary since it can be calculated by mining the service logs. However, all outgoing messages must carry time stamps for auditing purposes. The time stamp information is therefore removed from incoming messages and moved to outgoing messages.

These changes are bundled in one service redesign task with the goal to roll out a new version of the service as soon as possible. In particular, a new operation and wrapping port type for the synchronous communication is added to the WSDL of the service as shown in Fig. 4. receivePOSync reuses the same messages as its asynchronous counterpart receivePO in Fig. 1, taking into account the necessary modifications into the PODocument and POAck document types.

The simple sequence/receive set of BPEL activities in Fig. 2 is replaced by a pick activity that acts as a multiple option receive. Depending on whether the synchronous or asynchronous version of the operation is invoked, the appropriate response scheme is used (i.e. with a reply activity instead of the call back invocation for the synchronous part). These changes are depicted in Fig. 5.

While the redesign scenario also results in changes in the non-functional characteristics of POSERVICE, we will not consider them for purposes of simplifying the presentation.

IV. COMPATIBLE EVOLUTION OF SERVICES

In the previous section we discussed how existing works approach service evolution in terms of versioning and compat-

Fig. 3. POSERVICE WSDL – Service Improvement Scenario (PODocument only)

Fig. 4. POSERVICE WSDL – Service Redesign Scenario

Fig. 5. POSERVICE BPEL – Service Redesign Scenario

bility. Existing approaches use an informal notion of compatibility, relying on the empirical guidelines summarized in Table I. In the following sections, we provide a classification of the different aspects of service compatibility before formalizing its definition using type theory.

A. Aspects of Compatibility

For the purposes of this discussion we extend the definition for software components given in [20] and separate compatibility into two distinct dimensions: horizontal compatibility (or service interoperability) and vertical compatibility (or substitutability/replaceability). More specifically:

Definition 1: Horizontal compatibility or interoperability of two services expresses the fact that the services can participate successfully in an interaction as service provider and service consumer.

Horizontal compatibility is therefore a co-dependence relation between two interacting parties (services in the general
The underlying assumption is that there is at least one context under which the two services can fulfill their roles. Context in this case is a configuration of the environment in terms of the execution state of both service producer and service consumer, along with the status of their resources, and for a particular message exchange history. This assumption is implicit in the definition of the vertical dimension, and permeates all the definitions formal and informal that follow.

Definition 2: Vertical compatibility or substitutability (from the provider’s perspective) or replaceability (from the consumer’s perspective) of service versions expresses the requirements that allow the replacement of one version by another in a given context.

The combination of the two compatibility dimensions leads to the notion of T-shaped changes as depicted in Fig. 6. In the example of Fig. 6, overlapping hexagons denote compatible service versions. Service $S_1$ is horizontally compatible with $S_2$, meaning that $S_1$ interoperates with $S_2$ – either as a consumer or a provider or both. Similarly, $S_2$ is horizontally compatible with service $S_3$. There exist two more versions of service $S_2$ denoted by $S''_2$ and $S''''_2$, that are vertically compatible with each other (and horizontally compatible with $S'_2$) but incompatible with $S_2$ as denoted by the gap between $S_2$ and $S'_2$, $S''_2$. This signifies the existence of a major release of $S_2$ (namely $S'_{2}$), which broke the interoperability of $S_2$ with $S_1$ and $S_3$; $S''_2$ was then replaced by a minor release (i.e. $S''''_2$).

The two dimensions are therefore interrelated: substitutability and replaceability can be perceived as the property of preservation of interoperability for internalized changes to one or both of the interacting parties (producers or consumers). This enables referring simply to compatibility and denoting both aspects. If compatibility is achieved under all possible contexts, either on the vertical or the horizontal dimension, or both, then it is called strict substitutability/replaceability and interoperability, or strict compatibility, respectively.

B. Formal Definition of Service Compatibility

For the formalization of the compatibility between any two services we assume each service (version) is denoted by a description schema $S$ comprised of records $s$. Records represent the structural dependencies inside the service description using elements and their relationships as in [21], behavioral constraints in the form of behavioral contracts [22] and/or non-functional characteristics expressed as QoS properties [23] or dimensions [24].

Based on this assumption, we can take advantage of the existence of a subtyping relation that allows us to (partially) order different records based on their characteristics for defining compatibility. Subtyping allows us to check whether two records participate in a specialization/generalization relation and whether (under certain conditions that will be discussed later in this section) one record can replace another. We will be using the notation $s \leq s'$ to denote that record $s$ is a subtype of record $s'$, irrespective of whether $s$ is a structural, behavioral or non-functional record.

To formally define the compatibility of two service versions we first define a distribution of the set $S$ into two proper subsets $S_{pro}$ and $S_{req}$, representing the set of records for which the service acts as a producer and a consumer (of messages) respectively [25]:

Definition 3: $S_{pro}$ is the set of output-type records of a service description and $S_{req}$ is the set of input-type records.

Compatibility between service versions $S$ and $S'$ can be defined based on this distribution as follows:

Definition 4: Service Compatibility We define three cases of compatibility:

- Forward: $S <_f S' \iff \forall s \in S_{pro}, \exists s' \in S'_{pro}, s' \leq s$ (covariance of output).
- Backward: $S <_b S' \iff \forall s' \in S'_{req}, \exists s \in S_{req}, s \leq s'$ (contravariance of input).
- Full: $S < S' \iff S <_f S' \land S <_b S'$.

These definitions are in line with both traditional type theory and with the language-producing set-based theory proposed in [16]. Given the fact that the subset $S_{pro}$ represents the language produced by the service, then this definition of forward compatibility is equivalent to the informal definition given in the Background section. Furthermore, armed with this definition we can reason directly on new versions $S', S'', S''', \ldots$ of the service, comparing them on a record to record basis for checking their compatibility. The following sections build the necessary components for such a reasoning. We start with an abstract model for describing services, on which we define the subtyping relation between service records. From there we proceed by presenting an algorithm for checking the compatibility of services.

C. Service Description

In order to describe a service in our framework we use the notion of Abstract Service Descriptions (ASDs) we introduced in [21] and which we refine here. An ASD represents a particular version of a service and can be defined as the set of all its versioned records $S := \{s_i\}, i = 1, \ldots, N, j \in V$ where $V$ is the set containing the version identifiers vid for all records $s$. $S$ therefore contains one particular version for each of its constituent records.

Each ASD respects a particular meta-model (Fig. 7). This meta-model divides constructs in three layers: a structural, a behavioral and a non-functional layer. The essential characteristics of this meta-model can be found in meta-models for services description languages such as WSDL and BPEL.
elements generated by the concept and belong to a property domain. The messagePattern to be used for an operation is an example of a property domain, containing properties like One-way, Request-Response, etc. The property domains for each element are depicted as enumerations in Fig. 7.

- rel is the type of relation between the elements \((a, c, s)\) - aggregation, composition or association with the semantics defined in [21].
- mul is the multiplicity of the relationship, defined as \(mul := [\text{min}_{crd}, \text{max}_{crd}]\) where \(\text{min}_{crd}, \text{max}_{crd} \in \mathbb{N}\) (the set of natural numbers) is the minimum and maximum respectively multiplicities allowed for each member of the relationship, as denoted in Fig. 7.

For example, let’s assume the WSDL definition of the POService as shown in Fig. 1. For the purchase order document and its wrapping message we have elements PODocument and POMessage, generated by the ASD Meta-model concepts Information Type and Message, respectively:

\[
e_{pod} = (\text{name} = \text{PODocument}, \text{valueType} = \text{document}, \text{valueRange} = \{\text{N/A}\})
\]

\[
e_{m} = (\text{name} = \text{POMessage}, \text{role} = \text{input})
\]

(i.e., there are no attributes, valueType is 'document' and valueRange is undefined, and)

\[
(\text{as before}). \text{We can equivalently write these elements in shorthand notation as: } e_{pod} = (\text{PODocument}, \text{document}) \text{ and } e_{m} = (\text{POMessage}, \text{input}) \text{ respectively.}
\]

From the message schema of POService, the PODocument must contain exactly one order description item OrderInfo and one TimeStamp item, but it may contain one delivery description item Delivery Info. The respective elements ASD elements are \(e_{oi} = (\text{OrderInfo}, \text{string}), e_{ts} = (\text{TimeStamp}, \text{dateTime})\) and \(e_{di} = (\text{DeliveryInfo}, \text{string})\), and the multiplicities of the relationships between \(e_{m}\) and \(e_{oi}, e_{ts}, e_{di}\) elements must be \([1, 1], [1, 1]\) and \([0, 1]\) respectively. The relationships of the \(e_{pod}\) element are therefore written in this notation as:

\[
r(e_{pod}, e_{oi}) = (\text{name}_{s} = \text{PODocument}, \text{name}_{t} = \text{OrderInfo}, \text{rel} = a, \text{mul} = [1, 1])
\]

or in shorthand:

\[
r(e_{pod}, e_{oi}) = (\text{PODocument}, \text{OrderInfo}, a, [1, 1])
\]

\[
r(e_{pod}, e_{di}) = (\text{PODocument}, \text{DeliveryInfo}, a, [0, 1])
\]

\[
r(e_{pod}, e_{ts}) = (\text{PODocument}, \text{TimeStamp}, a, [1, 1])
\]

Elements \(e_{m}, e_{pod}, e_{oi}, e_{di}, e_{ts}\) and their relationships constitute the ASD representation of the POMessage data type in Fig. 1. Different versions of elements and relationships can be represented by referring to the set \(\mathcal{V}\) of all VIDs: \(e^{i}, r^{j}, i, j \in \mathcal{V}\) denote versions \(i\) and \(j\) of element \(e\) and relationship \(r\) respectively. In the following, we will simply write \(e^{i}\) or \(r^{i}\) as a shorthand for \(e^{i+k}, k \geq 1\) and \(r^{j+m}, m \geq 1\).
2) Behavioral Layer: The behavioral layer contains the records describing the perceived behavior of the service in terms of exchanges of messages grouped under service operations, and the conditions under which message exchanges may occur.

A number of different techniques have been proposed for describing and reasoning on the exchange of messages, such as business protocols based on finite state machines [27], [28] or deterministic finite automata [29], formal languages like TLA+ [30], communication action schemas [31], workflows [32], automata [33], timed protocols [34] and Calculus of Communicating Systems (CCS)-like constructs [22]. A notation for the behavioral description of services (called (behavioral) contracts) has been proposed in [22], which is conceptually very similar to our approach. For this reason we rely on that work for the definition of behavioral description and show how the necessary constructs for applying it are incorporated into our model.

Behavioral contracts $\sigma$ under [22] use three operators: continues with "+", external choice "⊕" and internal choice "∩". The behavioral contract $\sigma_1 = a_1.a_2$ means that $a_1$ is followed by action $a_2$. $\sigma_2 = a_1 + a_2$ signifies that the external party (the service client) chooses which action to perform ($a_1$ or $a_2$ but not both), whereas for $\sigma_3 = a_1 \oplus a_2$, it is the service that decides which action is to be performed. Furthermore, actions are distinguished as input (to the service), denoted by a simple action $a$, and output type (from the service to the client) actions, denoted by barred actions $\bar{a}$. If not specified explicitly it is assumed that an action can be either input or output type.

The ASD Meta-model in Fig. 7 contains the concepts Protocol (for behavioral contracts $\sigma$) and Activity (for actions $a_i$) and the stereotyped relation $sType$ with possible names follows, $eChoice$ and $iChoice$ corresponding to the operators ", + and $\oplus$, respectively. Activity defines a specific type of action to be performed on the basis of an operation (in the same manner as BPEL simple activities). The protocol described in Fig. 2 maps to the behavioral contract expression $\epsilon(\text{sequence}) = e_{	ext{ReceivePO}}.e_{	ext{SubmitPOAck}}$, which in turn corresponds to the ASD elements and relationships:

$\epsilon_{\text{sequence}} = (\text{sequence})$

$\epsilon_{\text{ReceivePO}} = (\text{ReceivePO}, \text{receive})$

$\epsilon_{\text{SubmitPOAck}} = (\text{SubmitPOAck}, \text{invoke})$

$r(\epsilon_{\text{sequence}}, e_{\text{ReceivePO}}) = (\text{sequence}, \text{ReceivePO}, \text{follows}, [1,1])$

$r(\epsilon_{\text{sequence}}, e_{\text{SubmitPOAck}}) = (\text{sequence}, \text{SubmitPOAck}, \text{follows}, [1,1])$

The expression above is the equivalent of Fig. 2 in ASD notation. In a similar fashion we can always map to a protocol $e_{prt}$ and its relationships to other protocols $r(e_{prt}, e_{prt})$ or activities $r(e_{prt}, e_{act})$ to the respective behavioral contract $\sigma(e_{prt})$.

Dealing with operational pre- and post-conditions to represent the conditions under which message exchanges can occur is more straightforward: we update the classic extension of behavioral specification by [36] and [37], that describe the behavior of an object in terms of pre- and post-conditions. The conditions are expressed as relatively simple logical expressions like $\text{pre.elems} \neq \{\}$ denoting a non-empty list of input elements. For the purposes of this discussion we will assume that these conditions are codified as groups of expressions that must be in a specific (boolean) status. The ASD Meta-model contains for this purpose the following concepts: Constraint elements to allow the definition of specific conditions to be satisfied, and Operation Conditions to group them and define whether they are to be used as pre- or post-conditions for protocols or operations. Operation Conditions and Constraint elements and their relationships are covered by the discussion on the structural layer since they use only basic relationships (association and composition).

3) Non-functional Layer: The non-functional layer consists of QoS and policy constraints in the forms of assertions that are associated with evolving services. In a similar manner to the behavioral layer, we overload the semantics of the layer elements and their relationships. Since our approach depends on the ordering of the service description records the remainder of this section will focus on ordinal QoS dimensions [24], i.e. QoS dimensions whose values can be ordered according to some predefined criteria.

More specifically, we adopt a simplified version of WS-Policy[12] for the description of QoS-related expressions. The concepts for these records and their property domains are depicted on the upper layer of Fig. 7. We assume that Assertions, containing statements about the acceptable and expected value ranges of ordinal QoS dimensions are organized by conjunctions or disjunctions into AssertionSets, grouped in turn as Profiles.

Using the ASD notation, an element of Assertion type $\text{e}_{\text{assert}}$ is a tuple 

$\text{e}_{\text{assert}} := (\text{name} : \text{string}, \text{dim} : \text{dimType} : \text{dimType}, \text{value})$

For the values of the non-functional profile of theOPSISERVICE for example, we have:

$\text{e}_{\text{assert1}} = (\text{assert1}, \text{availability}, \text{monotonic}, [80, 95])$

$\text{e}_{\text{assert2}} = (\text{assert2}, \text{latency}, \text{antitonic}, [15, 3])$

$\text{e}_{\text{assert3}} = (\text{assert3}, \text{performance}, \text{monotonic}, [90, 100])$

Each dimension $\text{dim}$ belongs to the DimentionTyple property domain denoting its behavior with respect to the ordering of its values. Monotonic dimensions order their values with increasing order; Antitonic order them in decreasing order. Availability values (a monotonic dimension) can be considered more general if they are higher, whereas response time values (an antitonic dimension) are more general the closer to 0 they are. Elements of Profile and AssertionSet type act as anchors for the relationships between the assertions. Expressing the grouping of assertions

under an assertion set (e.g. \(\text{aset}_1 = \text{assert}_1 \land \text{assert}_2 \land \text{assert}_3\)) in the example is denoted in a similar manner to the behavioral layer as:

\[
\begin{align*}
r(e_{\text{aset}_1}, e_{\text{assert}_1}) &= (\text{assert}_1, \text{assert}_1, \text{AND}, [1,1]) \\
r(e_{\text{aset}_1}, e_{\text{assert}_2}) &= (\text{assert}_1, \text{assert}_2, \text{AND}, [1,1]) \\
r(e_{\text{aset}_1}, e_{\text{assert}_3}) &= (\text{assert}_1, \text{assert}_3, \text{AND}, [1,1])
\end{align*}
\]

POSERVICE offers only one profile to its consumers \(e_{pfl}\), containing exactly one assertion set \(e_{\text{aset}_1}\):

\[
e_{pfl} = (pfl_1) \\
r(e_{pfl_1}, e_{\text{aset}_1}) = (pfl_1, e_{\text{asets}}, \text{OR}, [1,1]) \quad \text{(by convention)}
\]

Combinations of disjunctions and conjunctions and more complex logical expressions can be denoted in the ASD notation in a similar manner.

D. Subtyping of ASD Records

1) Structural Subtyping: By their definition, each element relationship are \textit{types} themselves and we can compare two elements or relationships by extending the subtyping relation as follows:

**Definition 6:** (Structural) Subtyping of elements and relationships

- For \(e = (\text{name}, \text{att}_1, \ldots, \text{att}_m, \text{pr}_1, \ldots, \text{pr}_n)\) and \(e' = (\text{name}', \text{att}'_1, \ldots, \text{att}'_m, \text{pr}'_1, \ldots, \text{pr}'_n)\) we define the sub-type relation between \(e\) and \(e'\) as:
  
  \[
e \leq e' \iff \text{name} \equiv \text{name}' \land \text{att} \leq \text{att}'_i, 1 \leq i \leq m \land \text{pr} \leq \text{pr}'_j, 1 \leq j \leq n
  \]

  that is, they have the same name identifier, and \(e'\) has less attributes and properties than \(e\), but the ones it has are more generic (super-types) of the respective attributes and properties of \(e\). By definition it holds that \(e = \emptyset \leq e'\).

- For \(r(e_s, e_t) = (\text{name}_s, \text{name}_t, \text{rel}, \text{mul})\) and \(r'(e'_s, e'_t) = (\text{name}'_s, \text{name}'_t, \text{rel}', \text{mul}')\) we define the sub-type relation between \(r\) and \(r'\) as:
  
  \[
r(e_s, e_t) \leq r'(e'_s, e'_t) \iff e_s \leq e'_s \land e_t \leq e'_t \land \text{rel} \equiv \text{rel}' \land \text{mul} \leq \text{mul}'
  \]

  that is, the elements \(e'_s, e'_t\) participating in the (new) relationship are super-types of \(e_s, e_t\) (and therefore \(\text{name}_s \equiv \text{name}'_s \land \text{name}_t \equiv \text{name}'_t\)) and the multiplicity domain of the relationship is a super-set of the respective one in the old relationship. We assume \(\emptyset \leq r'(e_s, e_t)\) iff \(\text{mul} = [0, N], \ N \geq 1\).

With respect to the property domains of Fig. 7 it holds that \(\text{int} \leq \text{double} \leq \text{string} \leq \text{document} \) for \text{DataType} and \(\text{one-way} \leq \text{request-response}, \text{notication} \leq \text{request-response} \) for \text{MessagePattern}. This allows us to modify not only the message payload but also the interaction protocol of the service operations under certain conditions that we discuss in the following. Both of these options are not allowed by the guidelines of Table I.

Consider for example the new version of POSERVICE from the Service Improvement Scenario as shown in Fig. 3 which requires the delivery information to be obligatorily submitted together with the purchase order (instead of optionally as in the previous version in Fig. 1, as indicated by the \text{minOccurs} = 0 attribute). \(r(e_{pod}, e_{di})\) is therefore replaced in the service description \(S'\) of the service by \(r'(e_{pod}, e_{di}) = (\text{PODocument, DeliveryInfo}, a, [1,1])\) and it holds that \(r'(e_{pod}, e_{di}) \leq r(e_{pod}, e_{di})\) since \(\text{mul}' \leq \text{mul}\), that is, \(r'(e_{pod}, e_{di})\) is a subtype of \(r(e_{pod}, e_{di})\). This should be expected since an optional data type in the message schema is more generic than the same message schema with the data type as mandatory.

2) Behavioral Subtyping: In [22] the authors introduce a behavioral subcontraction relation \(\leq\) between behavioral contracts. A behavioral contract \(\sigma\) is called a behavioral subcontraction of \(\sigma'\) that is \(\sigma \leq \sigma'\) if it manifests less interaction capabilities than \(\sigma'\). [22] contains the description and proofs required for checking whether this relation holds given \(\sigma\) and \(\sigma'\). During the presentation of the records of the behavioral layer we showed how to map protocol elements to their respective behavioral contracts. Applying therefore the subtyping relation and checking for compatibility between versions of elements in the behavioral layer is reduced to mapping them to the respective contracts and applying the subcontracting relation \(\leq\) between them. This is achieved by overloading the semantics of the subtyping relation for Protocol elements and adding the following condition in Definition 6:

\[
e_{pfl} \leq e'_{pfl} \iff \sigma(e_{pfl}) \leq \sigma(e'_{pfl})
\]

That is, elements of the Protocol type are mapped to their respective behavioral description and the subtyping check is performed in that formalism.

The addition of an option of \textit{synchronous communication} mode to the input of POSERVICE initiated by the Redesign Scenario from Section III for example, results in a protocol that is a supertype of the initial protocol of the service. We have shown above that the protocol of the initial version of the service maps to the behavioral contract \(\sigma(e_{sequence}) = a_{\text{ReceivePO}}, a_{\text{SubmitPOAck}}.\) The BPEL description of the POSERVICE in the Redesign Scenario (Fig. 5), is mapped to the ASD records:

\[
\begin{align*}
e'_{\text{pick}} &= (\text{pick}), \ e_{\text{seq}_1} = (\text{seq}_1), \ e'_{\text{seq}_2} = (\text{seq}_2) \\
e_{\text{ReceivePO}} &= (\text{ReceivePO}, \text{receive}) \\
e_{\text{SubmitPOAck}} &= (\text{SubmitPOAck}, \text{invoke}) \\
r(e_{\text{seq}_1}, e_{\text{ReceivePO}}) &= (\text{seq}_1, \text{ReceivePO}, \text{follows}, [1,1]) \\
r(e_{\text{seq}_1}, e_{\text{SubmitPOAck}}) &= (\text{seq}_1, \text{SubmitPOAck}, \text{follows}, [1,1]) \\
e'_{\text{ReceivePOS}} &= (\text{ReceivePOS}, \text{receive}) \\
e'_{\text{ReplyPOAck}} &= (\text{ReplyPOAck}, \text{reply}) \\
r(e'_{\text{seq}_2}, e'_{\text{ReceivePOS}}) &= (\text{seq}_2, \text{ReceivePOS}, \text{follows}, [1,1]) \\
r(e'_{\text{seq}_2}, e'_{\text{ReplyPOAck}}) &= (\text{seq}_2, \text{ReplyPOAck}, \text{follows}, [1,1])
\end{align*}
\]
\(r(e_{pick}', e_{seq2}) = (pick, seq_2, eChoice, [1, 1])\)

The equivalent expression in behavioral contract notation is \(\sigma'(e_{pick}) = (\text{aReceivePO, aSubmitPOAck}) + (\text{aReceivePOSync, aReplyPOAck})\) from which it can be seen that \(\sigma'(e_{sequence}) \preceq \sigma'(e_{pick})\) since \(\sigma'(e_{pick})\) contains \(\sigma'(e_{sequence})\) and allows for further interactions. Therefore, and according to the extension of Definition 6 we introduced, we can conclude that \(e_{sequence} \preceq e_{pick}'\). This means that a client that works with protocol \(e_{sequence}\) can also work normally with protocol \(e_{pick}'\).

Reasoning on the Operation Conditions and Constraint elements and their relationships, is sufficiently covered by Definition 6. Adding new Constraints \(e_{conj}' = (con_j, expression_j, true)\) and \(e_{opcon}' = (opcon, pre)\) for example, creates additional relationships \(r(e_{opcon}', e_{conj}') = (opcon, conj_j, c, [1, 1])\) and \(r(e_{opcon}', e_{conj}) = (opcon, conj_j, c, [1, 1])\) for which we know that \(r(e_{opcon}, e_{conj}) = 0\) \(\leq r(e_{opcon}, e_{conj})\) and \(r(e_{opcon}, e_{conj}) = 0\) \(\leq r(e_{opcon}, e_{conj})\) since \(e_{opcon} \neq 0 \land [1, 1] \neq [0, N], N \geq 1\).

3) Non-functional Subtyping: Extending the subtyping relation as defined in Definition 6 in the model of non-functional description we introduced requires two things: providing operators for ordering the value ranges for each assertion element with respect to how general/specific they are, and handling the special semantics of disjunctions and conjunctions. For the former, we base the ordering of assertions on the nature of their dimension and we use the relations already defined in Allen’s Interval Algebra [38] for relatively positioning intervals (here value ranges) on a dimension. For the latter we use the simple observation that an assertion set with more conjunctions is more restrictive (i.e. more specific) than one with less, while the reverse is true for disjunctions.

Definition 6 needs therefore to be supplemented by the conditions:

- \(e_{asrt} \leq e_{asrt} \iff \text{name} \equiv \text{name}' \land \text{dim} \equiv \text{dim}' \land v \leq v'\) with
- \(v \leq v' \iff \{ v\{=, <, s, f, m, o\}v' \text{ (monotonic dimensions)} \}
- \{ v\{=, >, f, si, mi, oi\}v' \text{ (antitonic dimensions)} \}

where the relations have the following semantics:
- \(=\) has equal length (and overlaps totally) with \(v'\),
- \(s\) \(v\) starts together with \(v'\) but finishes before it,
- \(f\) \(v\) starts after \(v'\) but it finishes together with it,
- \(m\) \(v\) meets \(v'\) at its finishing point (\(v\) finishes when \(v'\) starts),
- \(o\) \(v\) partially overlaps with \(v'\) – having started before \(v'\),
- \(<\) \(v\) takes place before \(v'\).

The inversions \(si, fi, mi, oi, >\) signify that the roles of \(v\) and \(v'\) are reversed (\(v > v'\) for example means that \(v'\) takes place before \(v\), etc.)

- \(\emptyset \leq r(e_s, e_t, OR, mul)\) and \(r(e_s, e_t, AND, mul) \leq \emptyset\)

The first addition expresses the ordering of values based on their monotonic or antitonic nature. More generic means extending the value range towards the direction that is considered “better”, allowing even for partial or no overlaps, provided that the range moves towards better values. The second addition formalizes the notion that the more options exist for a profile the more generic it is; the more conditions to be satisfied on the other hand in each profile, the more specific it is.

For the QoS characteristics of POSERVICE after the Service Improvement Scenario for example, we have in ASD notation the records:

\(e_{assert_2} = (assert_2, latency, antitonic, [0.075, .15])\)
\(e_{assert_3} = (assert_3, performance, monotonic, [81, 100])\)
\(e_{assert_4} = (assert_1), e_{pfl_1} = (pfl_1)\)
\(r(e_{assert_1}, e_{assert_2}) = (assert_1, assert_2, AND, [1, 1])\)
\(r(e_{assert_4}, e_{assert_3}) = (assert_1, assert_3, AND, [1, 1])\)
\(r(e_{pfl_1}, e_{assert_1}) = (pfl_1, assertion, OR, [1, 1])\)

By their definition, latency is an antitonic dimension (lower values are better) and performance is monotonic (the closer to 100%, the better), and we have:

- \(assert_2 = assert_2 \land latency = latency \land antitonic = antitonic\) \(\land ([.15, .3] \mi [.075, .15])\)
- \(assert_3 = assert_3 \land performance = performance \land monotonic = monotonic\) \(\land ([81, 100] \fi [90, 100])\)

From the extension of Definition 6 for non-functional elements we have \(e_{assert_2} \leq e_{assert_2} \land e_{assert_4} \leq e_{assert_3}\) (with \(e_{assert_1}\) remaining unchanged). These results correspond to the observation that a client that accepts latency between 0.15 and 0.3 seconds can also accept the more favorable latency between 0.075 and 0.15 seconds. In contrast, a client that expects performance between 90 and 100% cannot accept a performance that ranges between 81 and 100%, since this may result in performance below the acceptable lower limit (90%). Changing the performance of the service from 81 – 100% to 90 – 100% on the other hand is acceptable (for the client).

E. Reasoning on Service Evolution

Having established a type theory for all the layers of an ASD, it becomes possible to use the subtyping relation of ASD records to check for the compatibility of service versions. Reasoning about this decision is quite straightforward: by combining Definitions 4 and 6 (as extended accordingly for each layer) we can check whether both cases of compatibility are satisfied using the definition of subtyping for ASDs. The following sections discuss how this can be achieved.

1) T-Shaped Changes: We informally define the concept of T-shaped changes as the set of changes that respect service compatibility. We use three fundamental operators to describe the changes occurring to service descriptions: \(add\) for the addition of a record, \(del\) for the removal of a record and \(mod\) for the modification of the record (addition/removal of attributes or properties and so on). Combinations of these fundamental operators can be further used to produce more advanced operators like the renaming of a record. By applying these operators to an ASD \(S\) and for a record \(s\) we get the respective change
primitives: \textit{add}(s, S), \textit{del}(s, S) and \textit{mod}(s, S). Depending on
whether \(s\) is an element or a relationship, the change primitives
are expanded accordingly: \textit{add}(e, S) and \textit{add}(r(e_s, e_t), S) for
example signify the addition of an element \(e\) or a relationship
\(r(e_s, e_t)\) to \(S\), respectively.

The evolution of services rarely occurs in simple increments
and usually encompasses a number of changes to the service
description that occur simultaneously. For this reason we
define a \textit{change set} as the fundamental degree of change to a
service description:

\textit{Definition 7:} A change set \(\Delta S\) is a set of change primitives
\[
\Delta S := \{\text{operator}(s_i, S) | \text{operator} \in \{\text{add}, \text{del}, \text{mod}\}\}
\]
that when applied to an ASD \(S\) results in a new version of
the service \(S'\), signified by \(S' = S \triangleleft \Delta S\).

Versions of services can therefore be expressed in terms of
the change sets that are required for reconstructing a version
from a baseline (original) version, following the conventions
of SCM. For the purposes of this discussion we assume
that the change sets between ASDs are recorded during the
development of the services. If not, then the application of an
algorithm like the one presented in [39] could generate them.

We can classify change sets with respect to compatibility as:

\textit{Definition 8:} \textit{T-shaped changes} A change set \(\Delta S\) is called
\textit{T-shaped} iff, when applied to a service description \(S\), it results
in a fully compatible service description \(S' = S \triangleleft \Delta S\), that is,
\(S \triangleleft_c S'\) (using Definition 4).

The term “T-shaped change” refers to the relation between
the two aspects of compatibility as illustrated in Fig. 6. As
long as a change set \(\Delta S\) results in a horizontally or vertically
compatible (or both) version of a service, then it belongs to
the set \(\mathcal{T}\) of all possible T-shaped changes. Constraining
the evolution of services is therefore reduced to checking
\(\Delta S \in \mathcal{T}\).

2) \textit{Checking for Compatibility:} Reasoning on a change set
is performed in two steps:

- Distribution of the elements of \(S\) in sets \(S_{\text{pro}}\) and \(S_{\text{req}}\)
sets using Definition 3.

- Checking whether the change set is T-shaped or not using
Definition 8.

For the creation of the \(S_{\text{pro}}\) and \(S_{\text{req}}\) sets we initially select
all elements of input or output type in Fig. 7, starting with
elements like \textit{Messages}. Then, by taking advantage of the
relationships between elements in Fig. 7, we propagate this
property to all elements connected to them, following the
direction of the arrow of the relationship. Then, we “mark”
both the relationship and the connected element with the same
type (input or output) and we continue this process until there
are no more relationships to traverse. This way we construct
the \(S_{\text{pro}}\) and \(S_{\text{req}}\) sets, which contain:

- \(S_{\text{req}}\): 
  
  \begin{itemize}
  
  \item \textit{Message} elements with property value \textit{role}\textit{=input} and all \textit{Information Type} elements
  that are in an aggregation relationship with them, together with the respective relationships. \textit{Activity}
  elements with property \textit{act}\textit{=invoke} or \textit{act}\textit{=reply}. \textit{Operation Conditions}
  elements with property value \textit{role}\textit{=post} and all \textit{Constraint}
  elements that are in a composition relationship with them, together with the respective relationships.
  
  \end{itemize}

- \(S_{\text{pro}}\): 
  
  \begin{itemize}
  
  \item \textit{Message} elements with property value \textit{role}\textit{=output} or \textit{fault} and all \textit{Information Type} elements
  that are in an aggregation relationship with them, together with the respective relationships. \textit{Activity}
  elements with property \textit{act}\textit{=invoke} or \textit{act}\textit{=reply}. \textit{Operation Conditions}
  elements with property value \textit{role}\textit{=post} and all \textit{Constraint}
  elements that are in a composition relationship with them, together with the respective relationships.
  
  \end{itemize}
ΔS₃ [which corresponds to the guideline of adding optional message data types] is T-shaped, irrespective of whether the data types (represented by an `it` element) are added to a message that belongs to the provided or required set. This is because if it is the former case, then it does not affect the second step of CCF; if it is the latter case, then due to the fact that an optional relationship (with minimum multiplicity 0) is a super-type of the “empty” relationship by definition, and given that the rest of S remains unaffected, then it also passes the first step. ΔS₂ is also T-shaped under all cases following a similar reasoning.

ΔS₄ on the other hand is T-shaped only if the deleted operation has exclusively input messages and under the assumption that these messages can be ignored without affecting either the producer or the consumer. The respective guideline explicitly forbids this change set by being too conservative for the sake of safety. Our approach shows that such a modification to a service would not necessarily break existing consumers. If the receipt of the message is part of a larger communication protocol though, then this change set may not be T-shaped due to the respective constraints on the behavioral layer. Such cases should be handled with care.

ΔS₅ and ΔS₆ work in a different manner. They allow for flexible input messages and associated data types by allowing a more general multiplicity domain in their relationship. This implies that the service can accept more incoming messages or a wider message payload than before. They also restrict the output messages accordingly.

ΔS₇ accepts as T-shaped the addition of a non-optional data type to an output message and the removal of a message data type from input messages. The reasoning is the same as in the case of ΔS₃; as long as the consumer or the producer respectively can ignore the “additional” message payload, then the compatibility is preserved. Further T-shaped change sets can be generated in a similar fashion.

The set of T-shaped change sets that can be produced in an incremental way is therefore a super-set of the guidelines-based one in Table I. Enumerating all possible T-shaped change sets for all layers of service description, even by starting with a relatively simple meta-model as that of Fig. 7, is too lengthy of a process to be presented here and defeats the purpose of the framework. Nevertheless, if necessary or desired, it is shown that this process is feasible.

### B. Evolution Scenarios Revisited

The following section revisits the evolution scenarios defined in Section III and discusses if the changes they are proposing are T-shaped or not. The purpose of this discussion is to further illustrate our proposal and demonstrate the effect of each scenario to service developers and consumers.

#### 1) Service Improvement Scenario

This scenario results in changes in both the structural and non-functional layers of POSERVICE. More specifically and as discussed throughout the previous sections, S’ differs from S by:

- \( r'(e_{pod}, e_{di}) = (PODocument, DeliveryInfo, s, [1, 1]) \)
- \( e'_{asest} = (aset_1) \)
- \( e'_{pfl} = (pfl_1) \)
- \( r'(e'_{asest}, e_{assert_1}) = (aset_1, assert_1, AND, [1, 1]) \)
- \( r'(e'_{asest}, e_{assert_2}) = (aset_1, assert_2, AND, [1, 1]) \)
- \( r'(e'_{asest}, e_{assert_3}) = (aset_1, assert_3, AND, [1, 1]) \)
- \( r'(e'_{pfl}, e_{asest}) = (pfl_1, aset_1, OR, [1, 1]) \)

We have previously established that:
- \( r'(e_{pod}, e_{di}) \subseteq r(e_{pod}, e_{di}) \), with \( r(e_{pod}, e_{di}) \in S_{req} \) and \( r'(e_{pod}, e_{di}) \in S'_{req} \)
- \( e_{pfl_1} \not\subseteq e'_{asest} \) since \( e_{assert_3} \leq e_{assert_3} \Rightarrow e_{asest} \not\subseteq e'_{asest} \)

### TABLE II

**Patterns of Change Sets**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>T-shaped Change</th>
<th>Corresponding Guideline</th>
</tr>
</thead>
<tbody>
<tr>
<td>ΔS₁ = {add(it’, S), add((msg<code>, it’), S)\}, \( r(msg</code>, it’) = {msg_.name, it’.name, a, m}, ) ( m = [0, N], N &gt; 0 )</td>
<td>Yes, if ( msg_1 \in S_{pro} ) then there is no violation of covariance; if ( msg_1 \in S_{req} ) then it holds by definition that ( \emptyset \leq r(msg_1, it’) ).</td>
<td>Add (Optional) Message Data Types</td>
</tr>
<tr>
<td>ΔS₂ = {add(op’, S), add((op’, msg), S) } ∪ {add(op’, S), add(msg’, S), add((msg’, op’), S), ...}</td>
<td>Yes, reasoning in a similar fashion as above.</td>
<td>Add (New Operation)</td>
</tr>
<tr>
<td>ΔS₃ = {del(op, S), del((op, msg, it), S), ...}</td>
<td>No, if ( \exists j, msg_{i,j} \in S_{pro} ) due to covariance; Yes, otherwise(^a)</td>
<td>Remove Operation</td>
</tr>
<tr>
<td>ΔS₄ = {mod(op, S) } ∪ {mod((op, msg, it), S), ( r’(op, msg, it) = {..., mul’} )</td>
<td>Yes, if ( mul \subseteq mul’ \wedge msg_{i,j} \in S_{req} ) (contravariance) or ( mul’ \subseteq mul \wedge msg_{i,j} \in S_{pro} ) (covariance); No, otherwise.</td>
<td>Modify Operation</td>
</tr>
<tr>
<td>ΔS₅ = {mod(it, S) } ∪ {mod((it, it), S), ( r’(it, it) = {..., mul’} )</td>
<td>Yes, if ( mul \subseteq mul’ \wedge it_{i,j} \in S_{req} ) (contravariance) or ( mul’ \subseteq mul \wedge it_{i,j} \in S_{pro} ) (covariance); No, otherwise.</td>
<td>Modify Message Data Types</td>
</tr>
<tr>
<td>ΔS₆ = {add(it, S), add((msg, it’), S), ( r(msg, it’) = {msg_.name, it_.name, a, m}, ) ( m = [M, N], 0 &lt; M &lt; N )</td>
<td>Yes, iff ( msg_1 \in S_{pro} ) (covariance); No, for all other cases.</td>
<td>Add Mandatory Data Types</td>
</tr>
<tr>
<td>ΔS₇ = {del(it, S), del((it, msg, it), S), ...}</td>
<td>Yes, iff ( it_i \in S_{req} ) (contravariance); No for all other cases.</td>
<td>Remove Data Types</td>
</tr>
</tbody>
</table>

\(^a\)Referring to Table I, guidelines in italics are either new or not originally allowed.

\(^b\)i.e. an one-way operation for example can be deprecated without an effect on the consumer - the service can just ignore the incoming message.
By combining the above we find the change set \( \Delta S_{SIS} \) required by the scenario not to be T-shaped: the first step of CCF is violated since \( r'_1(e_{pod}, e_{ts}) \leq r(e_{pod}, e_{ts}) \) and \( e_{pfl} \not\subseteq e_{pfl}' \). In service versioning terms, this signifies the need for the creation of a major version of the service, requiring the consumers of POSERVICE to adapt or migrate to the new version.

This scenario illustrates the case for shallow changes: by trying to minimize errors and improve the service, the service developers unintentionally generate additional development effort for the service consumers. While this cost may appear small, it is impossible to predict the actual impact of such a change for service-based applications consuming the service. Furthermore, it has to be considered that the creation of the new version of POSERVICE must be accompanied by the execution of an appropriate decommissioning plan for the existing version to facilitate the transition to the new version (as discussed in Section II). This plan comes with additional costs in communicating the change to the consumers and running two active versions of the service (and their supporting implementation) in parallel for the transitional period. The costs of implementing the Service Improvement Scenario therefore may outweigh its benefits and in this case it has to be reconsidered.

2) Service Redesign Scenario: This scenario has two major effects on the POSERVICE: it changes its interaction protocol by replacing the simple sequence with a pick activity (adding a new operation to the structural description to support the additional entry point) and it also modifies the incoming and outgoing messages. We previously showed during the discussion on behavioral subtyping that \( e_{sequence} \leq e_{pick} \) and thus passes the first step of CCF. We only therefore have to check the records of the structural layer.

With respect to the (new) input and output messages, as shown in Fig. 4, the new ASD \( S' \) differs from the previous version by:

- the removal of the relationship \( r(e_{pod}, e_{ts}) = (PODocument, TimeStamp, a, \{1, 1\}) \).
- the addition of the relationship \( r(e'_{poack}, e_{ts}) = (POAck, TimeStamp, a, \{1, 1\}) \).

Since \( r(e_{pod}, e_{ts}) \not\subseteq S'_{req} \) and the rest of Information Type elements connected to \( e_{pod} \) remain unaffected, then all records \( s' \in S'_{req} \) pass the first step of CCF. The second step of CCF operates on all elements of \( S'_{pro} \), for all elements of which we can see that \( s' \leq s \) (since \( r(e'_{poack}, e_{ts}) \) does not appear in \( S'_{pro} \)) and therefore this step passes too.

Furthermore, the addition of the receivePOSync operation to the WSDL of the service depicted in Fig. 4 is mapped in ASD notation to the addition of element \( e'_{remsg} \) to \( S' \), together with its (structural) relationships \( r(e'_{remsg}, e_{msg}) \) and \( r(e'_{remsg}, e_{msgpack}) \) to the existing POMessage and POMessagesAck messages, respectively. In addition, the elements \( e'_{pick}, e'_{seq}, e'_{seq}' \) to the existing ReceivePOSync and ReplyPOAck and the respective relationships have to be added, and the \( e_{sequence} \) is to be removed and replaced by \( e_{pick} \). For these reasons it holds \( \Delta S_{SRS} \subseteq S'_{req} \) and \( e'_{ReplyPOAck} \subseteq S'_{pro} \). Using a similar reasoning as above we can see that CCF returns successfully with true and therefore \( \Delta S_{SRS} \in \mathbb{T} \).

In contrast to the Improvement Scenario, the Redesign Scenario is actually shallow. This means that the new version of POSERVICE \( S' \) can be implemented and deployed by replacing the previous version without any effect to existing consumers. Both new (using the synchronous communication capability) and old (using the asynchronous one) consumers can interact with the service using the same service interfaces. No particular decommissioning plan is necessary, and no additional costs (further than the development of the new service) are required. Being able to reason that the Redesign Scenario is T-shaped therefore guarantees that the effort and impact of implementing the change is minimum.

It has to be noted that using only the backward-compatibility preservation guidelines, summarized in Table I, service developers would not be able to decide on the compatibility of the new version produced by the scenario since no behavioral aspects are covered by the guidelines. Based exclusively on the structural layer, they would conclude the new version is not compatible since no modification of incoming and outgoing messages is allowed (other than the addition of optional elements). In the following sections we will focus on this divergence by investigating its roots and proposing specific solutions.

VI. IMPLEMENTATION

In order to demonstrate the efficacy and practicality of our approach, we performed a proof-of-concept implementation. Using this implementation, we can empirically validate our proposal in a controlled setting through a case study. In the following sections we discuss the technologies, methods and outcomes of these procedures.

For the implementation of our proposal, we used widely supported and open source tools. The resulting Service Representation Modeler (SRMod) prototype\(^\text{14}\) provides two key facilities required for the empirical validation of our proposal [40]: a graphical editor for defining ASD models of service versions, and a reasoning module that compares two ASD models and checks them for compatibility as discussed in the previous section. A high-level flowchart view of the architecture of the SRMod prototype is shown in Fig. 8. More specifically, service description versions (i.e. WSDL, BPEL and WS-Policy documents) are converted into ASD models by the model transforming capability of the graphical editor and then given as input to the reasoning module. The reasoning module uses CCF in order to check their compatibility and includes the results of this check into a report. This compatibility report is then returned to the graphical editor for visualization.

We developed the SRMod prototype as a plug-in for the Eclipse platform\(^\text{15}\). The first step for the implementation consisted of the definition of the meta-model to be used for the representation of the services in the prototype. For this purpose we used the bottom layer of the ASD Meta-model (Fig. 7). The various elements and relationships of the structural layer were encoded in the Emfatic language (part of the EMF Eclipse plug-in [41]). We also annotated the Emfatic specification of

\(^{14}\) Available at http://srmod.wordpress.com/

\(^{15}\) http://www.eclipse.org/
the meta-model with GMF-specific instructions. We then used the injection facilities of the Epsilon plug-in\(^\text{16}\) to convert the Emfatic specification of the meta-model into an ecore-type meta-model. Using this ecore meta-model, we automatically generated the Java code required for the graphical editor and the reasoning module. Fig. 9 shows an example of both functionalities in action.

The top part of the figure shows the ASD modle for one of the versions of the POSERVICE loaded in the graphical editor of the SRMod prototype. The editor provides the tools for creating and modifying a graphical representation of ASD models. It achieves this by offering a Palette panel (on the upper-right part of Fig. 9), which contains widgets corresponding to the various structural records in the ASD Meta-model. Selecting one of these widgets and pointing in the white canvas area adds the respective element to the ASD model. Adding or modifying the name, properties, relationships and the \(S_{\text{pro}}/S_{\text{req}}\) distribution (according to Definition 3) of the records is done through the Properties perspective (at the bottom of Fig. 9).

The reasoning module of the SRMod tool was implemented as a fully functional Epsilon program. We started by translating the CCF into a set of rules for records of the structural layer as follows:

\[
\begin{align*}
\text{op} \leq \text{op}' & \iff \text{name} = \text{name}' \\
\text{msg} \leq \text{msg}' & \iff \text{name} = \text{name}' \land \text{role} = \text{role}' \\
\text{r}(\text{op}, \text{msg}) \leq \text{r}'(\text{op}', \text{msg}') & \iff \text{op} \leq \text{op}' \\
& \land \text{msg} \leq \text{msg}' \land \text{mul} \subseteq \text{mul}' \\
& \ldots
\end{align*}
\]

This unrolling of the rules allowed us to encode the CCF in a straightforward manner and provide it as a module of the SRMod prototype. The module takes as input two ASD models and compares them, checking for compatibility as shown in Fig. 9. Currently, the results are returned in the Epsilon console and compares them, checking for compatibility as shown in Fig. 9. The reasoning module returned a false since an Information Type to Information Type relationship was found to violate the CCF.

If all checks are successful then the reasoner concludes with a true, otherwise it returns false. In the particular case the reasoning module returned a false since an Information Type to Information Type relationship was found to violate the CCF.

VII. Evaluation

Providing service developers with the means to control the evolution of services belongs conceptually to design science. As such, the evaluation of this work is performed along the lines of requirements for effective design science research. For this purpose we use the guidelines proposed in [42].

In particular, the utility, quality and efficacy of the proposed framework for the compatible evolution of services was evaluated in the previous sections using descriptive methods. Due to the nature of our approach, which combines theoretical with empirical aspects, we evaluated the design of our approach using the scenarios driven from the industrial case of the POSERVICE. In the previous sections, we demonstrated the impact of changes with varying complexity to the consumers of POSERVICE, describing how to avert consumer disruption through the application of the service compatibility theory. The proposed method of controlling service compatibility was compared and contrasted with existing approaches in Section V. We concluded that our approach by far covers and improves results of existing service evolution approaches and therefore our contribution is significant.

A. Empirical Validation

The prototype implementation of our proposal, described in Section VI, allowed us to design and execute a case study in order to validate our (theoretical) findings in a controlled environment of evolving services. The validation focused on the structural layer of services, working exclusively with WSDL descriptions, as typically used in practical settings.

More specifically, we modeled in the SRMod prototype the structural description of all versions of the POSERVICE discussed in Section III and checked them for compatibility. The results of this procedure agree with the theoretical results as presented in Section V: the Service Improvement Scenario was found not to be T-shaped, while the Redesign Scenario was concluded to be T-shaped as far as the structural layer is concerned – since the SRMod prototype is currently limited to checking for structural compatibility. Ongoing work is concentrating on providing the appropriate extensions to cover the rest of the layers of our approach.

For the empirical validation, we deployed the various versions of the POSERVICE in the Apache Axis2 Web services engine\(^\text{17}\), hosted in an Apache Tomcat servlet container\(^\text{18}\). We used the WSDL2Java code generation tool\(^\text{19}\) from the Axis2 toolkit to generate a skeleton of the service from the initial version of the service, to which we added the necessary business logic for implementing the functionality of the service. Developing a Service-Based Application (SBA) to act as the

\(^{16}\)http://www.eclipse.org/gmt/epsilon/

\(^{17}\)http://ws.apache.org/axis2/

\(^{18}\)http://tomcat.apache.org/

\(^{19}\)http://ws.apache.org/axis2/tools/1_4_1/CodeGenToolReference.html
client of the service was also achieved by using the same code
generation tool on the initial version of POSERVICE.

We then proceeded to use the generated SBA to invoke the
different versions of the deployed service. During run-time
we monitored the server and mined the client logs in order
to check whether each service version is invoked successfully,
or whether the service version (or the client) breaks. It was
observed that the service and the SBA broke in both scenarios,
signifying the incompatibility between the different versions
of the service. In the case of the Improvement Scenario this
was expected and confirms our theoretical findings. In the case
of the Redesign scenario however, this result diverges from
the theoretical prediction due to the inadequacy of the service
implementation technology to handle evolution in connection
with XML Schema validation.

This conclusion was derived on the basis of an exhaustive
investigation of this divergence between theory and practice.
In particular, we traced back the produced error messages in
order to locate the service container module that produced
them. Our findings confirm that the problem stems from the
processing of XML messages in both service provider and
client sides (see also [43]). This divergence manifests when
either side tries to validate the incoming and outgoing mes-
sages against an XML Schema that is no longer valid – but not
necessarily incompatible. This means that current XML-based
implementation technologies are unable to ignore dynamically
information not contained in the schema vocabulary. If both
parties could ignore the data types not in their message schema
and validate the rest of the message, then the empirical result
would be in agreement with the theoretical findings of this
work.

A work-around for enabling these types of changes is to
intercept the messages and apply to them an appropriate trans-
formation using a technology like XSLT\(^{20}\) or Schematron\(^{21}\) as
discussed in [43]. Nevertheless, it is our belief that this issue
is better handled on the level of service description languages
rather than building ad hoc work-arounds. In the following
section we explain how this issue can be handled in accordance
with the proposed theoretical framework.

B. Realization

To evaluate the ability of service description language
specifications to handle the mechanisms that support service
evolution as discussed in the previous sections we surveyed
their latest versions. In particular, we referred to the WSDL
2.0, BPEL 2.0 and WS-Policy 1.5 specifications. With the
exception of WSDL, all surveyed specifications do not contain
the notion of versioning. The WSDL 2.0 Primer\(^{22}\) briefly
discusses evolutionary strategies for evolving services, but in
a non-normative manner and by incorporating the strategies
found in [16].

As such, the WSDL 2.0 Primer concentrates on the guide-
lines for backward and forward compatibility as presented
in Table I. WSDL 2.0 is therefore more restrictive than our
approach with respect to service evolution. The authors of
the Primer however acknowledge that changes in the message
content depend on the type system used to describe them. The
weak typing approach taken in the processing of messages,

\(^{20}\) eXtensible Stylesheet Language Transformations (XSLT) Version 2.0
http://www.w3.org/TR/xslt20/

\(^{21}\) http://www.schematron.com/

\(^{22}\) WSDL Version 2.0 Part 0: Primer http://www.w3.org/TR/wsd120-primer
and the static binding of service and client implementations to WSDL documents leave little space for improvement given the limitations of existing technologies and standards for Web services.

In order to facilitate the realization of our compatible service evolution framework, we propose that the current WSDL specification works in tandem with BPEL & WS-Policy in order to integrate all aspects of service description into a tightly connected set of documents. While both BPEL and WS-Policy can currently refer to WSDL elements in their documents, the integration of the three languages is quite loose on purpose. As we have already discussed, and despite its market dominance, WSDL is limited in the amount of information that it can carry with respect to the needs of consumers. Providing a tighter integration of the three specifications into a vertically integrated document that combines all three specifications would make the serialization of the ASD model easier.

More importantly, a stronger typing system than the current one has to be used both on the level of XML processing and on the level of the respective standard specifications. The model of simple XML parsing backed by XML Schema validation, currently used in most Web services technologies, stiles evolution and creates unnecessary coupling in both service provider and consumer sides. Despite the option of XML extensibility, it is difficult to design for compatible service evolution without the possibility of ignoring the parts of a message that are not understood by the message consumer.

We therefore propose that the parsing and validation model should be replaced by an automatic marshaling of messages (that is, their transformation into the respective objects) and the check for compatibility on the level of records (using the CCF presented in Section IV). Static bindings should be replaced by dynamic bindings to interface classes. These classes are able to accommodate the subtyping of the messages and representations through the use of inheritance (in static languages like Java) or a combination of inheritance and dynamic binding of types (in dynamic languages like Ruby\(^2\)).

Finally, the use of XML namespaces for version identification should be replaced by (or be combined with) a more robust versioning mechanism. For instance, version attributes should be natively included in the service description document. While very practical and easy to implement, namespace-based techniques depend exclusively on the service developer to be realized as shown by our case study. This dependence increases the propensity for errors and miscommunication. Furthermore, using a different namespace identifier for each modification unnecessarily breaks the service clients and increases the maintenance costs by introducing additional versions. As such they should be used with caution.

VIII. RELATED WORK

Evolution is particularly important in distributed systems due to a complex web of software component interdependencies. As Bennet and Rajlich point out [3], attempting to apply conventional maintenance procedures (halt operation, edit source and re-execute) in large distributed systems (like

\(^2\)http://www.ruby-lang.org/en/
generation of adapters between service interfaces and implementations, based on the parametric transformation of the expected and the actually offered interfaces of the service. Interface adapters can also be layered on top of each other (e.g. in cross-stubs and custom handlers in [52] or chain of adapters in [53]) to “mask” the mismatches and maintain compatibility between providers and consumers. By using this technique, service developers deal with an (ideally) unique implementation endpoint that exposes multiple versions of interfaces, instead of multiple versions of the service. The maintenance cost then is moved to the consistency and efficiency of the layering of the adapters and out of the service life-cycle itself.

There are a number of issues with these corrective approaches with respect to service evolution. Firstly, adaptation does not necessarily happen in response to change; it may actually be the cause of change. For example, adaptation may be used for enabling the reuse of services (e.g. [31]). In this respect, adaptation is one of the means by which evolution manifests, the other being the replacement of the services used for the composition and the redeployment of a service in case of service compositions. Furthermore, the application of service adaptation techniques – both for interface and composition – is not always possible without explicit manual intervention. In this sense these approaches are limited in their automation. The required modifications may also interfere with the operation of other services by the same organization in terms of resources (computational and financial) and code. Service adapters avoid this risk by not requiring the redevelopment of the service. They however transfer the service adaptation cost to the effort required for developing, and more importantly, maintaining the adapters. Finally, the majority of the corrective approaches discussed above focus on the generation of the adaptation with the goal to automate the process without first checking whether the adaptation is necessary (in terms of compatibility). However, this is not always true as for example discussed in [52] and [53]. These approaches incorporate compatibility checks before attempting to generate suitable adapters.

Most of the approaches discussed in Section II can be classified as preventive approaches. As we explained in length, all these approaches take the very pragmatic road of providing a set of guidelines for the compatible evolution of services based on existing technologies. Of particular mention is the work of Becker et al. [17] that also presents an approach that constrains the evolution of services based on backward compatibility. They also however depend on a guideline-based approach which is limited in expressiveness and portability in other technologies. In this work, we are abstracting from the particular technology used for the implementation of services and present a theoretical framework that not only covers, extends and explains the outcome of these approaches, but also provides a formal foundation on which the effect of changes to the interface of service can be reasoned on. In [21] we discuss the fundamentals of the framework for compatible evolution, but we focus exclusively on the structural aspect of services. In this work we extend this framework to the behavioral and QoS-related aspects of service description and we update it to cover compatibility for those aspects accordingly.

Finally, as discussed in Section IV, our approach and all the approaches discussed in Section II assume that the change sets between service versions are available as part of the development process of the service. In case they are not, then a number of works on version differencing in UML models [54], MOF-based models [55] or business process models [39] can be used. However, these works focus on model consistency, and as such they do not do not provide a decidability theory for guaranteeing type safety and correct versioning transitions, so that previous clients can use a versioned service in a consistent manner. This is the very essence of the approach followed this paper.

IX. Conclusions & Future Work

In this paper we presented a theoretical framework and language-independent mechanisms to assist service developers in controlling and managing service changes in a uniform and consistent manner. For this purpose we distinguished between shallow (small-scale, localized) changes and deep (large-scale, cascading) changes and we focused on shallow changes. In particular, we provided a sound theory for service compatibility and reasoning mechanisms for delimiting the effect of changes which we keep local to and consistent with a service description.

The approach proposed in this paper is in contrast to the vast majority of existing approaches in the field. Most such approaches view service compatibility as strictly enforced by a set of empirical and technology-specific rules (e.g. WSDL-dependent guidelines) which indicate which changes are characterized as being compatible. This results in a very strict service evolution regime which prohibits potentially legitimate changes from being applied due to the limitations of current service technologies.

We presented a formally-backed compatible service evolution framework which is based on a technology-agnostic notation for the representation of services in the form of Abstract Service Descriptions (ASDs). ASDs act as a springboard to explain the versioning mechanisms for services. Using these results, we formally defined service compatibility and developed a theory for the compatible evolution of services. As part of this approach we introduced the notion of T-shaped changes, which enforce service compatibility between interrelated service versions in two dimensions: horizontally and vertically. We also demonstrated how to reason about the compatibility of service versions using the Compatibility Checking Function (CCF) in order to decide if changes to them are T-shaped or not. We validated our compatible service evolution framework in practice by means of a proof-of-concept prototype implementation in the form of the Service Representation Modeler (SRMod) tool and a case study. Based on the findings of this validation we provided a series of recommendations for the improvement of service description languages in the context of service evolution.

The compatible service evolution framework will be extended in the immediate future to show that the set of shallow changes is closed under compatibility-preserving change sets, and that our approach is complete in the mathematical sense.
by illustrating that the CCF can generate all possible T-shaped sets, as discussed in [35]. Following this, the SRMod prototype will be extended with appropriate behavioral and non-functional layer capabilities to ensure consistency in service versioning. While reasoning on the structural layer was sufficient for the purposes of this paper, extending this capability to the other layers is considered critical for the full implementation of the framework. Furthermore, the option to import ASD models directly from WSDL, BPEL and WS-Policy (currently performed semi-automatically) and to visualize the results of the compatibility check will also be added to the SRMod capabilities.

These changes will allow us to provide service developers with a comprehensive toolset for controlling the different aspects of service evolution. The application of the SRMod prototype in-the-field will allow us to draw useful conclusions about the efficacy of both the prototype and our work in general. Of course this process would also allow us to further improve and extend our research findings.

Finally, throughout all the work presented here, we have assumed a direct bilateral consumer/provider type of interaction between services and their clients. This introduces a certain amount of rigidity in service evolution. To relax this rigidity and allow for additional change scenarios that guarantee type-safety evolution we experimented with service contracting and Service Level Agreements (SLAs) [25]. In particular, we investigated the application of an intermediary construct in the form of a service contract interposed between service providers and consumers. A service contract can be used to represent a SLA between a service provider and consumer. Explicit contracts between providers and consumers allow for greater flexibility in evolving both parties in a compatible manner as they relax some of the assumptions regarding the ability of services to evolve while preserving their compatibility. We furthermore showed how even the contract itself can be a subject to change without affecting the interacting parties that it binds. Due to the fact that service contract formation depends on the subtyping relation as defined here, the work on contracting can be easily incorporated with the compatible evolution framework described in this paper.

ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

The authors wish to thank the reviewers for the many insightful comments and constructive criticism that have resulted in substantially improving the quality of this paper. The authors would also like to thank the members of ERISS, and especially Michael Parkin, Michele Mancioppi and Oktay Türetekten, as well as Barbara Pernici, Hossein Siadat and Mariagrazia Fugini at Politecnico di Milano for their feedback and support while writing and improving this paper. Many thanks also to Juan Vara and David Granada in the Kybele Research Group at the University Rey Juan Carlos for their invaluable help with the prototype implementation.

REFERENCES

A Programming Model for Self-Adaptive Open Enterprise Systems

Harald Psaier, Florian Skopik, Daniel Schall
Lukasz Juszczyk, Martin Treiber, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
Argentinierstraße 8/184-1, A-1040 Vienna, Austria
{lastname}@infosys.tuwien.ac.at

ABSTRACT

Open Web-based and social platforms dramatically influenced models for work. The emergence of service-oriented systems has paved the way for a new computing paradigm that not only applies to software services but also human actors. This work introduces a novel programming model for Open Enterprise Systems whose interactions are governed by dynamics. Compositions of humans and services often expose unexpected behavior because of sudden changes in load conditions or unresolved dependencies. We present a middleware for programming and adapting complex service-oriented systems. Our approach is based on monitoring and real-time intervention to regulate interactions based on behavior policies. A further challenge addressed by our approach is how to simulate and adapt behavior rules prior to deploy polices in the real system. We outline a testing approach to analyze and evaluate the behavior of services.

Categories and Subject Descriptors
C.2.4 [Distributed Systems]: Distributed applications; H.3.5 [Online Information Services]: Web-based Services; H.4 [Information Systems Applications]: Miscellaneous

General Terms
Design, Human Factors, Management, Performance

Keywords
Open Enterprise, Programming Model, Human Dynamics

1. INTRODUCTION

Service-oriented architecture (SOA) is an emerging paradigm to realize extensible large-scale systems. As interactions and compositions spanning multiple enterprises become increasingly commonplace, organizational boundaries appear to be diminishing in future service-oriented systems.

In such open and flexible enterprise environments, people contribute their capabilities in a service-oriented manner. We consider service-oriented systems based on two elementary building blocks: (i) software-based services (SBS), which are fully automated services and (ii) human-provided services (HPS) [13] for interfacing with people in a flexible service-oriented manner. Related efforts in service-oriented systems such as WS-HumanTask [10] attempt to model human interactions in top-down business processes assuming closed enterprise systems.

Here we discuss service-oriented environments wherein services can be added at any point in time. Following the open world assumption, humans actively shape the availability of HPSs. Without any coordination, such systems may exhibit undesirable properties due to unexpected behavior. Thus, social implications caused by human participation pose additional challenges to designing large-scale mixed SBS-HPS systems. However, with size also the effort for managing dynamically growing and loosely coupled systems is sharply increasing. Periodic adaptations are essential to keep a system within well-defined states, including stable load conditions or desired behavior. Due to the scale and inherent dynamics of open large-scale systems, new approaches for designing, developing and testing are required.

1.1 Motivating Scenario

Let us start with a service-oriented collaboration scenario that motivates our work. Today, processes in collaborative environments are not restricted to single companies only, but may span multiple organizations, sites, and partners. External consultants and third-party experts may be dynamically involved in certain steps of such processes. These actors perform assigned tasks with respect to prior negotiated agreements. Single task owners may consume services from external expert communities. For a single service consumer this scenario is shown in Figure 1.

Figure 1: Mixed Open Enterprise System.
We model a mixed Open Enterprise System (OES) consisting of HPSs [13] and SBSs that belong to different communities. The members of these communities are discovered based on their capabilities (see profiles) and main expertise areas (depicted as shaded areas), and are connected through certain relations (e.g., FOAF1). Community members receive requests from external service consumers, process them and respond with appropriate answers. A typical use case is the evaluation of experiment results and preparation of test reports in biology, physics, or computer science by third-party consultants (i.e., the OES). While the results of certain simple but often repeated experiments can be efficiently processed by SBSs, analyzing more complex data usually needs human assistance. For that purpose, HPS offers the advantage of loose coupling and flexible involvement of human experts in a service-oriented manner. Our environment uses standardized SOA infrastructures, relying on widely adopted standards, such as SOAP and the Web Service Description Language (WSDL), to unify humans and software services in one harmonized environment.

Failures and performance degradations may arise due to various reasons when operating an OES. For instance, performance degradations can be expected when a minority of distinguished experts become flooded with tasks while the majority remains idle. Such load distribution problems can be compensated by adapting the delegation behavior of actors as discussed in [14]. Furthermore, consider shifting interests and evolving skills of people. In that cases capabilities of HPSs as well as people’s interaction behavior may change over time, thus, requiring again adaptations of the underlying infrastructure.

1.2 Approach and Contributions

Our approach to ensure the smooth operation of mixed OESs makes use of several modules. The main contribution of this paper is the description of these modules, their modes of operation and discussions on major design decisions. We highlight the contributions of this work in Figure 2:

- **Open Enterprise System** hosts both SBSs and HPSs interacting to perform joint activities.

- **VieCure Middleware** provides a programming model including monitoring and adaptation mechanisms that control the OES and the simulation environment. Service policies to regulate behavior (interaction dynamics) are based on observations and control.

Adaptation strategies can be deployed in the Genesis Simulation Environment for testing purposes. Furthermore, logs, from either the simulation or the life OES can be analyzed to customize configurations and adaptation strategies.

- **Genesis Simulation Environment** allows to investigate the effects of policies and adaptation strategies.

**Paper Outline.** The remainder of this paper is structured as follows. Section 2 details our flexible adaptation approach and the prototype implementation of a novel middleware. Furthermore, we highlight the configuration management and the Genesis2 testbed generator framework. A discussion on collected experience and findings are provided in Section 3. We outline related work in Section 4 and conclude the paper in Section 5.

2. DESIGN AND ARCHITECTURE

In this section we introduce our programming model for adaptations in OES. After outlining the VieCure middleware’s components, we provide a sample environment specification for a OES in Genesis2’s programming language. This helps to highlight how our newly introduced programming model maps high-level policies to actions deployable to the environment. We conclude by outlining how different environments can be modeled, simulated, hosted, controlled and, of course, adapted with the programming model.

2.1 Middleware for Adaptation

At its core the middleware for adaptation provides a programming model for policy rules and actions which transparently adapt the OES underneath. The interface to the Configuration Manager offers monitoring and visualization features for the current OES structure. Additionally, it allows to adjust the middleware’s policies with rules and actions from a more high-level and goal-driven perspective.

![Figure 3: Middleware for adaptive OESs.](image)

The three-layer infrastructure of the VieCure Middleware is outlined in Figure 3. The top layer comprises the Configuration Manager. This is a tool-set to monitor, track, and analyze the OES structure. Additionally, by hiding the particulars of the programming model it allows to extend and change the entries of the middleware’s Policy Store easily and, thus, to influence the following adaptations.

The middle layer hosts the VieCure Middleware. It is organized according to a MAPE-K loop [6], the common design pattern used for self-adaptive systems. Therefore, it connects and forwards event information through the modules monitoring, analyzing, planning, and adaptation in loop-style. Different databases assist the analyses including

---

evaluation with event history. The loop connects the Logging Module to the Monitoring Interfaces. The data is converted to processable events in the Analysis Module. Events activate an examination of the rules in the Diagnosis and Planning module. The actions related to the triggered rules are then deployed to the OES by the Adaptation Module. All policies are defined by our Programming Model which provides a language specification to define the policy’s rules together with the actions. The following sections explain the concepts of the middleware and the surrounding layers in more detail.

2.2 Genesis WS Framework

The purpose of the Genesis2 framework (in short, G2) is to provide all means to host a SOA based environment including OESs. Originally designed to model Web services testbeds, we discuss G2 in the context of modeling, programming, and adapting OESs.

![Figure 4: Layered G2 topology.](image)

Figure 4 shows G2’s layered topology consisting of various elements. This layered topology enables software engineers to generate and control SOA environments. The top layer displays the runtime with the Generated WS Instances including services, clients, registries, etc. The next layer enables control over the generated instances. The WS Control layer comprises a comprehensive model of the deployed environment. It allows to steer the execution of the instances and propagates any model changes back to the environment. The following G2 Plugins layer allows to dynamically contribute external extensions. These could be environment specific and include, e.g., studied behavior models that are tested by the currently deployed environment.

G2 provides its own programming model and language. This is an extension of the Groovy script language with additional keywords and structures integrating the model to the language. Listing 1 gives a few examples. The model datatype allows to import an external complex data type with the method create(). The statement in line 3 shows the use of the webservice model to define an array of WS bodies with different properties and operations. Lines 25 and 26 demonstrate how a host is set up with the host model and then deployed with the webservice model’s deployAt() method. We will also refer to these models in our policy programming model presented next when we address the

Listing 1: Sample WS environment specification.

formulation of adaptive actions. For a more detailed description of G2 and its model we refer readers to [7, 16].

2.3 Programming Model for Policies

The VieCure middleware provides its own programming model for polices. Policies comprise rules and connected actions. In line with the event, condition, action paradigm, once an event triggers a rule because it matches the condition, e.g., threshold of a metric, an action is executed and an adaptation deployed to the environment. An event passed from the Analysis Module or manual intervention from the Configuration Manager activates rule evaluation.

In order to design a policy rule, we use JBoss Drools\(^3\). The structure of such a list of rules is presented in Listing 2. It illustrates three example rules matching the scenario of behavior adaptation outlined in Section 1.1. All rules are defined by a starting rule tag. The following when states the condition on which the rule fires. The then part defines the action that is deployed. By providing an array list multiple actions can be add and deployed simultaneously depending on the event type. In the example the first rule Overload deploys a suspend operation action if the workload or traffic to a specific service becomes intolerable. The next rule Deprec rated removes an operation if several previous invocation trials fail. Finally, the third rule Add extends the functionality of a service by adding a new operation. The last rule can be considered one that was added through the Configuration Management. In the present case the system designer, e.g., decided to extend a service with a new adaptation rule because new capabilities can be offered. The rule fires in any case and, thus, has an empty condition.

It is important to note, that all of the recovery actions in Listing 2 take a parameter c as input. This parameter represents a placeholder for a policy action definition. The

\(^3\)http://jboss.org/drools
the hosting environment. To give an insight on how adaptive actions can be created we provide in the following some samples that are executable on G2 hosts. G2 requires the parameter c to represent actions defined in G2’s Groovy-based programming language [7]. As pointed out, G2 provides model modification via the control layer. Changes on the model are automatically propagated to the running environment.

Listing 3 shows three example closures (code blocks) matching the input parameters of the previously stated adaptation rules in Listing 2. All closures presented access the webservice model introduced briefly earlier. Apart from defining Web services this model is also used to change the deployed instances of services at runtime. In the following cases the webservice model takes a input variable of any known type that can be used to filter particular service instances. Thus, the first closure definition of the model works as a filter for the affected instances. In the example cases the service definitions are selected by the variable name, e.g., provided by the triggering event. After changes are applied to the model, changes on the service models are deployed to the running environment at the end (method redeploy()).

In detail, the meaning of the adaptation actions stated in Listing 3 is the following. Rule Overload requires a temporary suspension of an operation. In the corresponding action we assume that diagnosis recognized that operation sendResult() (c.f., Listing 1) causes overloads to the service because it fills the result document queue. The action definition from line 4-9 in Listing 3 states that the affected services’ operation is overwritten by a simple code that returns a suspension message. For the next rule Deprecated the same operation is taken offline. The related action code shows in line 14 another webservice model method to get a distinct operation of the service model and in line 15 a method to remove the deprecated method.

Finally, for the Add rule in line 20 we require another model of the G2 programming model. The same as with

---

Listing 2: Policy rules with recovery actions.

define two adaptation actions, need to be simulated with the help of previously presented and defined policies, on varying environment conditions. As the motivation explains, we are in particular interested in service environments including human actors. In such simulations human behavior including making choices, taking decisions, working on tasks, or performing actions, need to be simulated with the help of previously observed and examined log data.

The G2 programming model introduced in Section 2.2 ap-
plies the concept of closures to equip services with individual behavior. Referring to lines 9, 14 and 28 in the sample script of Listing 1 it can be recognized that dStrat is defined as a global closure in the service model. While in the example the running service expects an executable algorithm resulting in an accept or reject of a received document the strategy can be changed at any time during runtime by setting a new closure content to the global placeholder (line 28). Also, with this method the strategy can be set individually for the otherwise identical service definition. This makes the simulation more authentic. Furthermore, once VieCure’s adaptation loop is activated, adaptation strategies are deployed to the simulated environment and conclusions on the effectiveness of adaptation strategies can be made.

2.5 Configuration Management

Figure 5 shows screenshots of the Web-based configuration management tool and an example FOAF profile that can be retrieved from the Web application.

The purpose of the network visualization view as shown by Figure 5(a) is to analyze complex behavior in OES environments. The view is based on a graph structure modeled as $G = (V, E)$ where $V$ represents the set of services (HPS and SBS) and $E$ the set of edges based on interactions. The network view is obtained by mapping raw SOAP-interactions into a graph representation composed of nodes (services) and edges (interaction links). The users access information captured from the OES (Figure 5). In our implementation, this is performed by selecting a particular set of logs which are associated with an Experiment ID. By default, the collaboration network is visualized as a graph view as depicted in Figure 5(a). The user is able to select a particular metric (in this case trust) threshold by moving a slider bar. A reduced metric threshold results in more target nodes and edges being added to the visualization. Interactions can be retrieved as FOAF profiles (see Figure 5(b)) that include <foaf:interest> tags. FOAF-based network profiles are especially useful for analyzing relations using semantic reasoning engines. Also RDF-based query languages such as SPARQL\(^5\) can be used to query network data. This mechanism can be used to retrieve and aggregate captured profiles from distributed environments (e.g., from multiple instances of the logging service).

3. DISCUSSION AND FINDINGS

OESs pose a number of new challenges with their flexibility and their tendency to unexpected behavior. Todays SOA infrastructures can provide the necessary flexibility, with easily operable interfaces and interaction channels enabling communication and collaboration. However, those do not provide means to handle unpredictable changes or system degradations. With humans collaborating in such networks, there are usually no preplanned top-down composition models. This results in changing interaction and behavior patterns that possibly contradict and at times result in faults from varying conditions and misbehavior in the network. A conflict resolution management for such an infrastructure must not only monitor events but also react

\(^{5}\)http://www.w3.org/TR/rdf-sparql-query/
situation-dependent according to rules. For this purpose, the VieCure middleware provides two interfaces for high flexibility. One towards the system, with the capabilities to gather log data and send adaptive actions. The other offers monitoring information and configuration handles for environment management above the middleware. In order to be consistent with self-adaptive methodologies, VieCure implements a loop-style log data transformation from event to adaptive action through the filter of the analysis modules and the policies of diagnosis. One of the major contributions of this work is the policy programming model for VieCure’s Policy Store. The presented model allows the Configuration Manager to state rules that are triggered on events. Rules contain placeholder for the actions. The implementation of the actions depends on the action interface of the environment. In the presented study, actions are stated in the programming model language of the test environment. The reason is to test the created rules previous to deployment to a real environment.

4. RELATED WORK

Two main research directions on self-adaptive properties emerged in the past years. One initiated by IBM and presented by the research of autonomic computing [15] and the other manifested by the research on self-adaptive systems [12]. Whilst autonomic computing includes research on all possible system layers and an alignment of self-* properties to all available system parts, self-adaptive system research pursues a more global and general approach. The efforts in this area focus primarily on research above the middleware layer and consider self-* methodologies that adapt the system as a whole. Regarding runtime evaluation, several approaches have been developed which could be applied for testing adaptation mechanisms. SOABench [3] and PUPPET [2], for instance, support the creation of mock-up services in order to test workflows. However, these prototypes are restricted to emulating non-functional properties (QoS) and cannot be enhanced with programmable behavior. By using Genesis2 [7], which allows to extend testbeds with plugins, we are able to implement scenarios which behave flexible enough to test diverse adaptation mechanisms [11]. Human-Provided Services [13] close the gap between SBS and humans desiring to provide their skills and expertise as a service in a collaborative process. Instead of a strict predefined process flow [9], these systems are denoted by ad-hoc contribution requests and loosely structured collaborations. The required flexibility induces even more unpredictable system properties responsible for various faults. The contributed middleware for self-adaptation and testing approach enables the recovery by restricting, for example, delegation paths or establishing new connections between services. The availability of rich and plentiful data on human interactions in social networks has closed an important loop [8], allowing one to model social phenomena and to use these models in the design of new computing applications such as crowdsourcing techniques [4].

5. CONCLUSION AND OUTLOOK

The main objective of this work was to introduce a middleware with adaptation features based on a novel programming model. We illustrated the adaptation of complex interactions in OESs using the programming model.

In future work we plan to evaluate the adaptation framework with various kinds of collaboration networks including recent crowdsourcing and other distributed problem-solving platforms. It will then also become essential to distribute and duplicate some of the components of the adaptation framework, e.g., logging, diagnosis and analysis modules.

Acknowledgment

This work received funding from the EU FP7 program under the agreements 215483 (SCube) and 215175 (COMPAS).

6. REFERENCES

Managing Evolving Services
Michael P. Papazoglou, Vasilios Andrikopoulos and Salima Benbernou

Services are subject to constant change and variation, leading to a continuous service re-design and improvement effort. However, service changes should not be disruptive by requiring radical modifications in the very fabric of services or in the way that business is conducted.

In [1], we defined the term service evolution as the continuous process of development of a service through a series of consistent and unambiguous changes. The evolution of a service is expressed through the creation and decommissioning of different service versions during its lifetime. These versions must be aligned with each other in a non-disruptive manner and in a way that allows a service designer to track modifications and their effects on the service. To control service evolution, a designer needs to know why a change was made, what its implications are, and whether the change is consistent. Eliminating spurious results and inconsistencies that occur due to uncontrolled changes is a necessary condition for services to evolve gracefully, ensure stability and handle variability in their behavior.

We can classify the nature of service changes depending on their causal effects as:

- **Shallow changes**: these are small-scale incremental changes that are localized to a service or are restricted to the clients of that service.
- **Deep changes**: these are large-scale transformational changes cascading beyond the clients of a service, possibly to entire value chains (end-to-end service networks).

Typical shallow changes focus on structural level changes (service types, messages, interfaces and operations) and business protocol changes (the conversations in which the service participates). Typical deep changes include policy induced (pertaining to business agreements between service providers and consumers), operational behavior (i.e., the service fulfilling its expected function in a timely and orderly manner) and non-functional changes (relating to QoS issues and Service Level Agreement guarantees for individual and end-to-end services).

While both shallow and deep changes need an appropriate versioning strategy, deep changes furthermore introduce several intricacies of their own, and require the assistance of a change-oriented service life cycle to allow services to react appropriately to changes as they occur. In this article we discuss a causal model of service changes that addresses the effects of both shallow and deep changes. In particular, we describe a formal approach to deal with shallow changes and a versioning strategy to support multiple versions of services. To address the problems that occur due to deep changes we introduce a change-oriented service life cycle methodology and describe its phases. In order to facilitate the discussion we first compare approaches in software and component evolution.
This article is largely based on concepts and definitions found in [1]. The definitions used in this article have been revised and amended on the basis of formalization and compatibility analysis, prototype implementation, comparison with functionality offered by open standards and an empirical in-depth investigation using an industrial strength case study, which we present in this article.

**Background**

Evolution in software systems has been traditionally considered as either a part, or a synonym of *software maintenance*. The insight gained by early studies resulted in empirical laws that drive and govern the evolution of software systems [2]. Evolution is particularly important in distributed systems due to a complex web of software interdependencies. As Bennet and Rajlich point out [3], attempting to apply the conventional maintenance procedure (halt operation, edit source and re-execute) in large distributed systems (like the ones emerging in service-oriented environments) is not sensible. The difficulty of identifying which software artifacts form the system itself is non-trivial, especially in the context of large service networks. In addition, the lack of ownership and access to the actual source code (if any) of third-party services (due to the Service Oriented Architecture (SOA) principles of encapsulation and loose coupling), does not allow the application of many of the maintenance techniques like refactoring or impact analysis.

The difficulty of approaching the evolution of systems as a purely maintenance activity has already appeared in the study of component engineering in general, and component evolution in particular. The basic ideas and solutions for the evolution of Component-Based Systems (CBS) are summarized in [4]. Evolving a component includes changes in both its *interfaces* and its *implementation*, with each one of these aspects having different evolutionary requirements. Due to their composability and emphasis on reuse, components exhibit strong dependencies with the components they consume. Changing a component may therefore have implications for other components, and upgrading to a new component may require for both versions (old and new) to be deployed in parallel while the transition takes place. Finally, identifying and distinguishing between different versions of components require the introduction of Software Configuration Management (SCM) techniques like version identifiers incorporated into the component metadata. Since version identifiers do not explain what changes occurred between versions, checking for compatibility between versions has to be performed separately.
While evolving, both components and services may offer multiple interfaces for the same functionality. This makes it possible to sustain several simultaneous versions. There are, however, subtle differences between these two related technologies summarized along the following dimensions:

- **Type of Coupling**: Services make use of abstract message definitions to mediate their binding with respect to each other. They focus on message and event definitions rather than method signatures, which typify components [5].

- **Type of Invocation**: Services introduce the concept of service capability, which describes the classification, functionality and conditions under which a particular service can be discovered and invoked. This leads to reactive services (which can respond to environmental demands without compromising operational efficiency). Components focus typically on locating and invoking other components by name.

- **Type of Binding**: An SOA application may choose a service dynamically on the basis of QoS, using parameters such as response time, throughput, availability, etc. Components depend on mechanisms like glue-coding, wrappers, delegation or aggregation for binding [5].

- **Type of Composition**: Service composition leads naturally to the creation of higher-level services which are typically long running coordinated workflow service arrangements specified according to business protocols. Component composition is typically on a lower level that depends on the component model used [6].

Historically and conceptually, CBS can be considered as a predecessor of Service Oriented Computing. However, it has to be kept in mind that components and services differ in terms of coupling, binding, granularity, delivery and communication mechanisms and overall architecture (see “Services vs. Component Evolution” sidebar).

**Case Study**

To provide a systematic way of looking at service evolution, we use the industrial-strength Automotive Purchase Order Processing case study developed jointly with IBM Almaden and used as one of the validation scenarios in the S-Cube Network of Excellence (http://www.s-cube-network.eu/). The Automotive Purchase Order Processing case study is an example of how to realize standardized supply-chain activities using SOA-based processes for a fictitious enterprise in the automobile industry called Automobile Incorporation (AutoInc). AutoInc consists of different business units, e.g. Sales, Logistics, Manufacturing, etc., and collaborates with external partners like suppliers, banks and transport carriers. The case study describes a typical automobile ordering process, where customers can place automated orders with AutoInc.

Within this context we performed a case study on the effect of changes to different services. We developed various evolutionary scenarios, describing the required actions and proposed modifications to purchase order processing services. These scenarios are expressed either as maintenance actions (e.g., optimization of the performance of some services) or as reengineering efforts (complete re-design of service interfaces) and are described in the sections that follow.
Dealing with Shallow Changes

Shallow changes affect both individual and end-to-end services. To deal with shallow changes we discuss some helpful practices for service compatibility and versioning derived at the basis of structural and business protocol service changes.

Service Version Compatibility

To be able to deal with message exchanges between a service provider and a service client we require that they still exchange valid messages despite interface changes that may happen to either side. To achieve this, we must rely on the notion of service version compatibility. Service version compatibility guarantees that we can introduce a new version of either a provider or a client of service messages without changing the other. Compatibility can be classified under two dimensions [7]:

- **Horizontal compatibility** or interoperability of two services. This expresses the fact that services can participate successfully in an interaction, either as service producers or service consumers.
- **Vertical compatibility** or substitutability (from the provider's perspective) or replaceability (from the consumer's perspective) of service versions. This expresses the requirements that allow the replacement of one version of a service by another in a given context.

There are two types of changes to a service definition that can guarantee version compatibility [8]:

- **Backward compatibility**: a new version of a message client is introduced and the message providers are unaffected. The client may introduce new features but should still be able to support all the old ones.
- **Forward compatibility**: a new version of a message provider is introduced and the message clients that are only aware of the original version are unaffected. The provider may have new features but should not add them in a way that breaks any old clients.

Some types of changes that are both backward- and forward-compatible include for example the addition of new service operations to an existing service description. In this case we are talking about (full) compatibility. Full compatibility allows the replacement of an existing service version with an equivalent (that is, compatible) one without any effect on the correct operation and the performance of its clients.

From a practical standpoint, compatible service evolution in the services description standard WSDL 2.0 is limited to service changes that are either backward or forward compatible, or both [9]. The types of service changes that are compatible are:

1. Addition of new WSDL operations to an existing WSDL document.
2. Addition of new XML schema types within a WSDL document that are not contained within previously existing types.
Incompatible change types on the other hand include: removing an operation, renaming an operation, changing the parameters (in data type or order) of an operation and changing the structure of a complex data type. In [7], an alternative approach is discussed for enabling the compatible evolution of services. Instead of restricting service changes to the short list above, a theoretical framework is presented which allows for reasoning on the evolution of services. As a result, further compatible changes (called T-shaped) are allowed; for example, removing data elements from incoming message data types and adding data elements in outgoing message data types. Service version compatibility for structural changes is based on two fundamental premises of type theory [10]:

- **Service argument contra-variance**: if we redefine the argument of a service, the new argument types must always be an extension (generalization) of the original ones.
- **Service result co-variance**: if we redefine the result of a service, the new result types must always be a restriction (specialization) of the original ones.

For example, the message payload of services in the AutoInc case study can be enhanced with time-stamping information if the messages are produced as output of the service, but not if they are consumed as input from the service. This reasoning is inverted when considering the evolution of the clients of the AutoInc services. In order to realize these capabilities, the existing model of validating incoming messages against their original (and possibly obsolete) XML schema has to be replaced by a direct marshalling of the message content into objects – and the compatibility of the message with respect to the expected message schema has to be checked separately [7].

When evolving a business protocol, states and transitions may be added to or removed from an active protocol. A new version of a protocol is created each time its internal structure or external behavior changes. The perception that clients have of a specific protocol is called a **protocol view**. Since the client's view of a protocol is restricted only to the parts of the protocol that directly involve the client, a client might have equivalent views on different protocols. Clients whose views on the original and target protocols are the same are not affected by evolution. Business protocol evolution is considered for example in [11] where the authors distinguish between various aspects of protocol changes. **Protocol compatibility**, aims at assessing whether two protocols can interact, i.e., if it is possible to have a conversation between the two services despite changes to their protocols. Compatibility of two protocols can be either **complete**, i.e., all conversations of one protocol can be understood by the other protocol, or **partial**, when there is at least one conversation possible between the two protocols.

Protocol compatibility is essential for re-designing service interfaces while allowing existing clients to consume them. In the case of the AutoInc, it allows for the replacement of single entry point asynchronous interfaces with more complicated multiple-entry interfaces that provide for both synchronous and asynchronous communication on demand. This in turn allows for the deployment of one service interface to cover both types of communication instead of having to provision for multiple service interfaces, as we describe below.
Versioning Shallow Changes

A robust versioning strategy is needed to allow for upgrades and improvements to be made to a service, while continuously supporting previously released versions. Service versioning is therefore an important issue for service developers and providers alike, building on the notion of service version compatibility as discussed above. In the case of service evolution, the cost of provisioning for multiple service interfaces is non-linear. As with component evolution, the development of a new interface requires additional effort in binding the interface versions with the underlying implementation(s). In the case of multiple active service versions however, each active version additionally requires access to a number of resources in the supporting infrastructure in order to be enacted. Furthermore, each version adds managerial overhead in terms of monitoring and auditing in order to ensure that it complies with the agreed-upon SLAs. As such, providing for multiple interfaces in services can be overtaxing for the service provider. The emphasis is therefore put on minimizing the amount of active versions through the use of compatible changes.

Figure 1 Service Versioning

With a compatible change, the service implementation need only support the latest version of a service interface [12], for example, implementation version 1.1 of a Receive Purchase Order service in AutoInc, supporting interface version 1.1 in Figure 1. A client may continue to use a previous service version (interface 1.0) while adjusting to a new version of the interface description at a time of its choosing or until the version is decommissioned. With an incompatible change, the client receives a new version of the interface description (interface version 2.0 in Figure 1) and is expected to adjust to the new interface
before the old interface is decommissioned. Either the service will need to continue to support both versions of the interface during the hand-over period (having one active and one deprecated version as in Figure 1), or the service and the clients are coordinated to change at the same time. An alternative is for the client to continue until it encounters an error, at which point it uses the new version of the interface. While this is common practice, it is prone to errors and inconsistencies in the interaction of providers and consumers and for this reason it should be avoided.

**Dealing with Deep Changes**

Deep changes characterize complex services and require that such services be redefined and possibly realigned within an entire end-to-end service. Deep service changes require a change-oriented service life cycle methodology to provide a sound foundation for spreading changes in an orderly fashion such that impacted services in a service-chain are appropriately (re-)configured, aligned and controlled as the changes occur.

The purpose of the change-oriented service life cycle is to ensure that standardized methods and procedures are used for the efficient and prompt handling of all service changes, in order to minimize the impact of change-related incidents upon service operation and quality. This means that in addition to functional (structural and behavioural) changes, a change-oriented service life cycle must deal with policy induced, operational behaviour and non-functional changes. The objective is to achieve actual end-to-end QoS capabilities for end-to-end services to achieve the proper levels of service required, by ensuring that services are performing as desired and that out-of-control or out-of-specification conditions are anticipated and responded to appropriately. This includes traditional QoS capabilities, e.g., security, availability, accessibility, integrity and transactionality, as well as service volumes (e.g., number of service events, number of items consumed, service revenue) and velocities (i.e., its performance characteristics). The combination of these measurements provides all the information needed to understand how an enterprise is performing in terms of its services.

Figure 2 illustrates a deep change-oriented service life cycle that comprises a set of inter-related phases, activities and tasks that define the change process from the start through to completion. Each phase produces a major deliverable that contributes towards achieving change objectives. Logical breaks are also provided in the change process and are associated with key decision points. The phases of the life cycle are discussed in the following.
The initial phase “Need to Evolve” focuses on identifying the need for change and scoping its extent. One of the major elements of this phase is understanding the causes of the need for change and their potential implications. For instance, compliance to regulations is a major force for change. This may lead to the transformation of all services within a service network. Here, the impacted individual services in an end-to-end service (or services-in-scope) need to be identified. In addition, service performance metrics, such as Key Performance Indicators (KPIs), need to be collected. In the case of the AutoInc Purchase Order Processing service for example, a large percentage of the response time of the public (partner-exposed) services was attributed to the communication overhead between services of different business units. A re-design of the services of AutoInc is therefore required, which triggers the next phase of the life-cycle.

The second phase in Figure 2 (“Analyze impact of changes”) focuses on the actual analysis, re-design or improvement of existing services. The ultimate objective of service change analysis is to provide an in-depth understanding of the functionality, scope, reuse, and granularity of services that are identified for
change. The problem lies in determining the difference between existing and future service functionality. To analyze and assess the impact of changes, organizations rely on the existence of an “as-is” and a “to-be” service model, rather than applying the changes directly on operational services. Analysts rely on an “as-is” service model to understand the portfolio of available services. This model is used as basis for conducting a thorough re-engineering analysis of the current portfolio of available services that need to evolve. The “to-be” service model is used as basis for describing the target service functionality and performance levels after applying the required changes.

To determine the differences between these two models a gap analysis model is used to help prioritize, improve and measure the impact of service changes. Gap analysis is a technique that purports a services realization strategy by incrementally adding more implementation details to an existing service to bridge the gap between the “as-is” and “to-be” service models. Gap analysis commences with comparing the “as-is” with the “to-be” service functionality to determine differences in terms of service performance (KPI measures) and capabilities. Service capabilities determine whether a process is able to meet specifications, customer requirements, or product tolerances. The gap analysis in the case of AutoInc revealed that, despite that many existing internal to the process services could be reused, composite services depending on services in different units of AutoInc had to be redrawn. It also resulted to proposed changes to public services, potentially impacting the clients of the end-to-end service. The resulting analysis however showed that the “to-be” model would perform better while respecting all existing SLAs.

As service changes may spill over to other services in a service chain, one of the determining factors in service change analysis is the ability to recognize the scope of changes and functionality that is essentially self-sufficient for the purposes of an evolving service. When dealing with deep service changes, problems of overlapping or conflicting functionality, several types of problems need to be addressed [13]:

- **Service flow problems**: Typical problems include problems with the logical completeness of a service upgrade, problems with sequencing and duplication of activities, decision making problems and lack of service measures.
- **Service control problems**: Service controls define or constrain how a service is performed. Broadly speaking there are two general types of control problems: problems with policies and business rules, and problems with external services.
- **Overlapping services functionality**: In such cases a service-in-scope may (partially) share identical business logic and rules with other related services. Here, there is a need to rationalize services and determine the proper level of service commonality. During this procedure, service design principles [14] such as service coupling and cohesion need to be employed to achieve the desired effects.
- **Conflicting services functionality** (including bottlenecks/constraints in the service value stream). Conflicts include problems where a service-in-scope is not aligned to the business strategy, a service pursues a strategy that is in conflict with, or is incompatible with the value chain of which
it is a part, and cases where the introduction of a new policy or regulation would make it impossible for the service-in-scope to function.

- **Service input and output problems**: These problems include problems where the quality of service input or output is low, and timeliness input or output problems where the needed inputs/outputs are not produced when they are needed.

Cost estimation involves identifying and weighing all services to be re-engineered to estimate the cost of the re-engineering project. In cases where costs are prohibitive for an in-house implementation, an outsourcing policy might be pursued. In the case of AutoInc, cost estimation showed that the implementation of the “to-be” model would come at an acceptable price. Nevertheless, it was also concluded that a part of the cost of the migration was in reality mitigated to the end-to-end to service clients since they needed to adapt to the new services’ design. To this effect it was deemed appropriate to modify the service charging policies.

During the third and final phase (“Align Refine Define” in Figure 2), all of the new services are aligned, integrated, simulated and tested and then, when ready, they are put into production. To achieve this, a service integration model is created to facilitate the implementation of the service integration strategy. The service integration model establishes, among other things, integration relationships between service consumers and providers involved in business interactions. It also includes steps that determine message distribution needs, delivery-responsible parties, and provides a service delivery map. Finally, the service integration model is concerned with message and process orchestration needs. The resulting service integration strategy includes service design models, policies, SOA governance options, and, reliance on organizational and industry best practices. All these concerns need to be considered when designing integrated end-to-end services that span organizational boundaries.

The role of the services integration model ends when a new (upgraded) service architecture is completely expressed and validated against technological specifications provided by infrastructure, management/monitoring and technical utility services. For the AutoInc case study, the services integration model resulted into a comprehensive plan for restructuring and reorganizing the services constituting the offered end-to-end service which was shown to be more effective and cost-efficient than the previous one.

**Summary & Future Work**

In the previous sections we proposed a causal model for addressing service changes that deals with the effects of both shallow and deep changes. In the case of shallow changes we defined a theoretical approach to decide whether a change is shallow and a versioning strategy to support multiple versions of services.

To address the problems of deep changes we introduced a change-oriented service life cycle methodology and described its phases. In particular, we discussed when a change in a service is triggered, how to analyze its impact and what are the possible implications of the implementation of the change for
the service provider and consumers. Due to its wide scope and the multitude of issues related to the change-oriented life cycle further research on this subject is essential. A formal model for deep changes, on the basis of the one for shallow changes is the main goal of our future work. Of particular interest is the transition between the two formal models (from shallow and deep changes) and the way they handle changes according to contractual service specifications [15].

References


Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

Bios

Michael P. Papazoglou is the Professor of Computer Science at Tilburg University. He is the Scientific Director of the European Research Institute in Service Science (ERISS) and the EC's Network of Excellence, S-Cube. He is also an honorary professor at the University of Trento in Italy, and professorial fellow at the Universities Lyon 1.
(France), Univ. of New South Wales (Australia) and Universidad Rey Juan Carlos, Madrid (Spain). Prior to this he was full Professor and head of School of Information Systems at the Queensland Univ. of Technology.

His research interests lie in the areas of service oriented computing, web services, large scale data sharing, business process management and federated information systems and distributed computing. He is a Golden Core Member and a Distinguished Visitor of the Institute of the IEEE Computer Science Section.

**Vasilios Andrikopoulos** works as a post-doctoral researcher for the European Research Institute in Service Science (ERISS) in Tilburg University, Netherlands. He received his Ph.D. from the department of Information Management in Tilburg University, Netherlands.

His research interests include service-oriented architecture and computing, with an emphasis on the evolution of services.

**Salima Benbernou** is full Professor of Computer Science at Paris Descartes University, France. She has published widely in highly reputable journals and conferences, and has been a Program Committee member of several conferences. She has participated in a number of French and EU-funded research projects, being coordinator in some of them.

Her research interests include formal models for service oriented computing, privacy in information systems and databases, data sharing and knowledge representation.
Using a Lifecycle Model for Developing and Running Highly Interactive Distributed Applications

D. Meiländer¹, S. Gorlatch¹, C. Cappiello², V. Mazza², and R. Kazhamiakin³

¹University of Muenster (Germany), ²Politecnico di Milano (Italy)
³Fondazione Bruno Kessler (Italy)

Abstract. We describe a case study on using the generic Lifecycle Model developed in the S-Cube project for a novel class of Real-time Online Interactive Applications (ROIA), which include, e.g., massively-multiplayer online games, distributed simulations, e-learning and training. We present how the Lifecycle Model addresses the specific challenges of ROIA services: large number of concurrent users connecting to a single application instance, frequent real-time user interactions, negotiation and enforcement of Quality of Service (QoS) parameters, adaptivity to changing loads, and competition-oriented interaction between users, other actors, and services. We describe the implementation aspects of the development and the adaptation phases of the Lifecycle Model, and report experimental results on a sample online game application.

Keywords: Service-Oriented Architecture, Service Engineering, Real-Time Online Interactive Applications, Scalability

1 Introduction

Service-oriented applications are developed for constantly changing environments with the expectation that they will evolve over time. Many service-oriented system engineering (SOSE) methodologies have been proposed aiming at providing approaches, methods and (sometimes) tools for researchers and practitioners to engineer service-oriented systems. SOSE methodologies are more complex than traditional software engineering (TSE) methodologies, having to deal with new challenges while keeping the principles of TSE. The additional complexity results mainly from open world assumptions, co-existence of many stakeholders with conflicting requirements and the demand of adaptable systems. A number of service lifecycle models have been proposed by both industry and academia. However, none of the proposed models has either reached a sufficient level of maturity or been able to fully express the aspects peculiar to SOSE. The S-Cube project [1] combines existing techniques and methodologies from TSE and SOSE to improve the process through which service systems will be developed.

This paper describes an industrial-strength case study for the S-Cube Lifecycle Model in the emerging and challenging area of Real-time Online Interactive Applications (ROIA) which include such popular and socially important applications.
as multi-player online computer games, high-performance simulations, e-learning, etc. There are several new challenges in these applications: thousands of simultaneous user connections to one application instance, optimized allocation of resources, and monitoring and steering to maintain the QoS parameters in dynamic distributed environments. Within the European edutain@grid project [2], a service-oriented architecture was designed which focuses on the main challenges of ROIA.

The paper studies how the application of the S-Cube Lifecycle Model to the applications on top of the edutain@grid architecture enables the designer to identify suitable adaptation mechanisms and design patterns for the challenging area of ROIA. We briefly introduce the S-Cube Lifecycle Model for SOSE in Section 2, followed by a description of the edutain@grid architecture for ROIA in Section 3. We describe the application of the Lifecycle Model on the case study scenario from edutain@grid in Section 4 and report experimental results of a sample ROIA in Section 5.

2 The Lifecycle Model for Service-Oriented Applications

The S-Cube Lifecycle Model for adaptable Service Based Applications (SBAs) (see Figure 1) comprises two main cycles: (i) a typical design-time iteration cycle that leads to the explicit re-design of the application in order to adapt it to new needs (i.e., evolution), and (ii) an adaptation cycle at runtime that can be used when the adaptation needs are addressed on-the-fly. The two cycles coexist and support each other during the lifetime of an application [3].

![Figure 1: The S-Cube Lifecycle Model.](image-url)

The development of an SBA starts with the former cycle that inherits some common aspects from the traditional software application lifecycle but is modified in order to deal with specific adaptation issues. In case of ROIA, already in the Requirements Engineering & Design phase it is necessary to identify monitoring and
adaptation requirements to guarantee high update rates and to design specific monitoring and adaptation methods.

The *Construction* phase of an SBA is often performed in the form of a service composition. The construction could be manual (service integrator defines an executable process composed of concrete and abstract services), model-driven (service orchestration models are generated by abstract models) or automated (starting from service models, the executable SBA is automatically generated). In the domain of ROIA, it is necessary to implement suitable parallelization, adaptation and scalability mechanisms. Then, after the *Deployment and Provisioning* phase in which the ROIA application is introduced to customers, the *Operation and Management* phase relies on the monitoring activities that use the designed monitored properties to derive the status of the application and detect changes in the context or in the system that could require adaptation or evolution. In fact, starting from this phase, ROIA developers can decide to execute the right hand side of the lifecycle if an evolution of the application is required (in terms of redesigning the application which is conducted offline and makes the application temporarily unavailable to customers), or otherwise the ROIA is managed online by enacting adaptation actions at runtime (executing the left hand side). E.g., an iteration of the evolution cycle may become necessary to protect the application against new attacking methods (by introducing suitable monitoring parameters) or to introduce new QoS requirements.

In the adaptation cycle, it is important to define the adaptation needs that can be caused by: changes in the functional and non-functional aspects, e.g., unreliable hoster resources cannot preserve QoS requirements, or changes of the context in which the application is running, e.g., increasing user numbers in the evening hours creating peak load. In the domain of ROIA, adaptation mechanisms need to be proactive and transparent to users in order to adapt the application during runtime.

### 3 A Service-Oriented Architecture for ROIA

In this section, we describe the specific features of Real-time Online Interactive Applications (ROIA) and express their major design and execution aspects in the context of the S-Cube Lifecycle Model from Section 2. ROIA pose many new challenges for SOSE including: large number of concurrent users connecting to a single application instance, frequent real-time user interactions, negotiation and enforcement of precise Quality of Service (QoS) parameters, adaptivity to changing loads and level of user interaction, and competition-oriented interaction between users, other actors, and services.

Within the edutain@grid project, a distributed service-oriented architecture (see Figure 2) was implemented that is based on the interaction of four main actors [7]: (1) *End-user* accesses ROIA sessions through graphical clients, typically purchased as a DVD; (2) *Scheduler* negotiates on behalf of the end-user appropriate ROIA sessions based on the QoS requirements (e.g. connection latency, application domain, friends, expertise); (3) *Hoster* is an organisation that provides a computational and network infrastructure for running ROIA servers; (4) *Resource broker* provides a mechanism
for application Schedulers and Hosters (and possibly other actors) to find each other in a large-scale environment and negotiate QoS relationships.

Figure 2: edutain@grid architecture for ROIA.

In the following, we describe the edutain@grid platform (Figure 2) using the application area of online gaming, and identify suitable services for the implementation of the adaptation cycle of the S-Cube Lifecycle Model (Figure 1).

**Scheduling Service.** The Scheduler receives from the user QoS requirements which can be performance-related (e.g. maximum allowed latency, minimum bandwidth) or game-specific (e.g. game genre, minimum number of players) and negotiates with existing Hosters that best fit these requirements. The result is a contract for the interaction of the end-user with the application. The mapping of players to game servers, as well as the allocation of Hoster resources to game servers, takes place as a distributed negotiation between the Scheduler and Hosters. The result of the negotiation is a performance contract that the Scheduler offers to the end-user and which does not necessarily match the original QoS request. The user has the option to accept the contract and connect to the proposed session, or reject it.

**Runtime Steering Service.** During the game session, situations may occur which affect the performance, such that the negotiated contracts are impossible to be further maintained. Typical perturbing factors include external load on shared resources, or overloaded servers due to an unexpected concentration of users in “hot spots”. The steering service interacts at runtime with the monitoring service of each Hoster for preserving the negotiated QoS parameters for the duration of the session. Following an event-action paradigm, a violation of a QoS parameter triggers appropriate adaptive steering or rescheduling using the API of the real-time layer. Thereby the Runtime Steering Service contributes to the “Identify Adaptation Needs” and “Identify Adaptation Strategy” phases of the adaptation circle of the Lifecycle Model.
**Resource Allocation Service.** Typically, each Hoster owns a Resource Allocation service responsible for allocating local resources to the clients. This service receives from the Scheduler connection requests formulated in terms of QoS requirements such as minimum latency or maximum bandwidth, and returns a positive answer if it can accommodate them. Online games are characterized by a large number of users that share the same application instance and interact across different game servers. The atomic resource allocation units for users are, therefore, no longer coarse-grain processes, but rather fine-grained threads and memory objects, which are obviously harder to address and more sensitive to external perturbations.

**Capacity Planning Service.** The load of a game session depends heavily on internal events, e.g. interactions of avatars. Also external events may occur, such as the user fluctuation over the day or week with peak hours in the early evening [5]. Hence, it becomes crucial for a Hoster to anticipate the future game load. The Capacity Planning Service predicts future load of Hoster resources by employing neural networks [7] and thereby contributes to the “Identify Adaptation Needs” and “Identify Adaptation Strategy” phases of the adaptation circle of the S-Cube Lifecycle Model.

The real-time layer of edutain@grid is implemented by the *Real-Time Framework (RTF)* [7] that distributes game state calculations among the participating servers. For this purpose, RTF provides efficient parallelisation and adaptation concepts and implements suitable monitoring capabilities, which we explain in the following.

**Real-Time Adaptation Mechanisms.** To enable the service scalability to a high number and density of users, RTF distributes game sessions adaptively, based on several adaptation strategies illustrated in Figure 3. **Zoning** [4] distributes the game world into disjoint zones, each zone being assigned to one server. **Instancing** uses multiple, independently processed copies of highly frequented subareas of the game world, each copy being processed by one server: if a user moves into one frequented subarea, he is assigned to one of the available copies. **Replication** [6] assigns multiple servers to a single game world zone with a high load. The responsibility of computing the entities' states in that zone is divided equally among the assigned processors.

RTF’s adaptation strategies are used during the “Enact Adaptation” phase of the adaptation circle of the Lifecycle Model as explained in Section 4.

**Monitoring Service** observes the QoS parameters negotiated by the Hoster as performance contracts which must be preserved throughout the game session. Several monitoring parameters are summarized in particular profiles which support monitoring of low-level QoS parameters, as well as of game-related metrics (like entity positions, messages sent or received, or end-user activity information such as idle, spectator, or active) crucial for guaranteeing an adequate game experience to the users. RTF's monitoring services are used by the steering and capacity planning services to observe QoS and as input for load prediction using neural networks.
In this section, we demonstrate how the Lifecycle Model of Section 2 is applied to ROIA applications by exploiting the edutain@grid architecture of Section 3. The lifecycle is based on various adaptation- and monitoring-specific actions and related design artifacts. The main aspects for designing an adaptive application are: the application requirements, the adaptation strategies, and the adaptation triggers.

Our analysis of ROIA applications has identified the application requirements in Table 1. Besides the functional and non-functional application requirements already described in Section 3, at this stage, it is also important to identify the requirements to be considered for the design of adaptation actions. In the case of ROIA, it is especially important that the mechanisms implementing the monitoring and adaptation infrastructure are non-intrusive, i.e., they take place in parallel with the normal execution of the application without stopping or blocking it and users are not aware of changes happening inside the application. Since the considered system must guarantee the QoS requirements, proactive adaptation must be supported in order to prevent QoS violations.

<table>
<thead>
<tr>
<th>Functional Requirements</th>
<th>- Correct execution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Non Functional Requirements</strong></td>
<td><strong>Client related requirements</strong></td>
</tr>
<tr>
<td></td>
<td>- Short response time (&lt; 100 ms)</td>
</tr>
<tr>
<td></td>
<td>- Frequent interactions between users</td>
</tr>
<tr>
<td></td>
<td><strong>Game session-related requirements</strong></td>
</tr>
<tr>
<td></td>
<td>- Resources use for a variable # of users</td>
</tr>
<tr>
<td></td>
<td>- Suitable parallelization concepts</td>
</tr>
<tr>
<td></td>
<td>- High number of concurrent users in a single application instance</td>
</tr>
<tr>
<td></td>
<td>- High update rate (5-100 updates/s)</td>
</tr>
</tbody>
</table>
Adaptation Requirements
- Transparency of the adaptation
- Instance-specific adaptation during runtime
- Need for proactive adaptation
- Autonomy (self-adaptability)
- Efficiency of adaptation actions

Table 1: Application requirements for ROIA.

The application requirements drive the selection of the adaptation strategies. According to the Lifecycle Model, we identify the following five adaptation strategies for applications on top of the edutain@grid architecture:

1. **User migration**: Users are migrated transparently from one server to another server (without allocating new resources) which is replicating the same zone. The migration of users is triggered if there are underutilized resources that can be used for processing the replicated zones of overloaded resources.

2. **Zoning**: New zones are added during runtime to the virtual environment which are assigned to additional game servers. Zoning provides the best scalability of all adaptation strategies but is not transparent to users (since the geography of the virtual world is changed), so it is only used for high numbers of users that cannot participate otherwise.

3. **Replication**: New game servers are added transparently during runtime in order to increase computation power for highly frequented zones. When replicating a zone, a number of users are migrated to the replica to initiate the workload distribution. However, replication implies an additional inter-server communication and thus, its scalability is limited. To address the demand for autonomic and self-adaptable adaptation strategies, the number of active replicas for a particular zone is monitored to decide whether activating additional replicas is feasible.

4. **Instancing**: Creates a copy of a zone which is processed by a different server than the original zone. Since users in different copies of the same zone cannot interact with each other, replication is generally preferred to instancing for ROIA in order to support high interactivity between users. However, instancing as an adaptation strategy is useful if the overhead of replication would be too high.

5. **QoS negotiation** with several distributed hosters which includes (i) adaptation of existing contracts, or (ii) negotiation of new contracts with new hosters. Typical scenarios include the usage of stronger resources of the same hoster (QoS adaptation) or leasing cheaper resources from a new hoster. Thereby, QoS adaptation might be an alternative to replication by allocating more powerful resources to serve more users. Since QoS adaptation needs a longer time, replication and instancing provide better scalability, but QoS adaptation can be used to overcome small peaks in resource shortage.

For designing adaptable SBAs, adaptation strategies must be related to adaptation triggers. Adaptation triggers and suitable trigger rules are defined considering all scenarios at runtime in which application requirements may be violated. For ROIA, adaptation triggers are mainly related to changes in Service Quality, in the computational context and in users’ requirements, unexpected increment of the users’ accesses, and specific users’ needs. In the edutain@grid architecture described in Section 3 proactive adaptation is planned on the basis of prediction values from the capacity planning service. For example, load balancing may anticipate increasing user
numbers in the evening hours and request appropriate resources. Then, the adaptation trigger (predicted increase in user numbers) is related to the change in the context (i.e., time period).

The list of adaptation triggers provides to the application designers the variables to be monitored at runtime and thereby drives the design of the monitoring mechanism. In ROIA, monitored properties include CPU or memory load on hoster resources, the number of concurrent users in an application instance, incoming/outgoing bandwidth capacity on a host, average packet loss, and connection latency.

By considering the application requirements and the adaptation strategies, we distinguish the following scenarios for triggering adaptation in ROIA applications:

1. Change in Quality of Service: QoS violations which were not expected nor predicted, e.g., caused by unreliable hoster resources. In this scenario, user migration, replication or instancing is used for adaptation in order to overcome performance bottlenecks as fast as possible. We decide which of the three adaptation mechanisms to use in a particular situation depending on the amount of free resources and number of active replicas: in order to minimize costs for the application provider, migrating users to underloaded resources is preferred if the additional load can be already compensated by running resources; otherwise replication is preferred to instancing in order to support a high level of interactivity between users; if activating additional replicas is not feasible since the communication overhead would be too high instancing is used for adaptation.

2. Change in computational context: a change in the computational costs of calculating the application state, e.g., caused by increasingly or decreasingly frequent interactions between users. To prevent QoS violations, adaptations should be enacted as quickly as possible using one of the adaptation strategies: user migration, replication or instancing (depending on free resources and number of replicas).

3. Change in business context: a change in user preferences which was not predicted in advance, e.g., many new users connecting to the application. In this scenario, user migration, replication or instancing (depending on the amount of free resources and number of replicas) are used for adaptation.

4. Prediction values from the capacity planning service: The edutain@grid capacity planning service (described in Section 3) gathers information about the users’ preferences and triggers adaptation proactively and autonomously. Depending on the predicted number of additional users, the strategies like QoS negotiation, replication or adding new zones during runtime are considered for adaptation since predicted adaptations can be planned ahead. If the maximum number of replicas is already reached, then instancing might be chosen as an alternative.

Table 2 shows how the described adaptation triggers and adaptation strategies are linked together, and provides examples for trigger rules and monitored values:

<table>
<thead>
<tr>
<th>Adaptation Trigger</th>
<th>Monitored variable</th>
<th>Adaptation Trigger rule</th>
<th>Adaptation Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change in Quality of Service</td>
<td>response time, throughput, resource usage, average packet loss, connection latency, update rate, service availability</td>
<td>update rate &lt; 25 updates/s</td>
<td>user migration, replication or instancing</td>
</tr>
</tbody>
</table>
We observe that for each adaptation trigger, various adaptation strategies may be suitable. The ROIA application is realized in the *Construction* phase of the S-Cube Lifecycle Model in order to include all the monitoring features and technical infrastructure needed for proactive adaptation. The autonomy required by the application has a strong impact on the design and realization of the monitoring mechanisms. Since ROIA applications are very dynamic, not all the failures/critical situations are identified at design time. Therefore, there is a need to continuously update the knowledge about the behavior of the system, e.g., in order to improve prediction values for proactive adaptation a continuous training of neural networks based on the previous outcomes is used to support an effective design of the adaptive application [7].

After the construction phase, the application is deployed, i.e. it is ready to be executed and managed. During the *Operation and Management* phase, the application is running and all the previously designed adaptation triggers are monitored to detect changes in the context or in the system that could require adaptation or evolution. Starting from this phase, the ROIA application is managed online by enacting adaptation actions during runtime; if evolution is required, the right hand side of the lifecycle, is executed again. E.g., evolution of a particular ROIA might be necessary if the application is facing new attacking mechanisms by fraudulent users or in case of changing user requirements (e.g., new QoS parameters are introduced), since there may be a need to define additional sensors and monitors, as well as to change adaptation selection strategies (e.g., strategy for balancing/distribution). In these cases the evolution phase of the Lifecycle Model is iterated again and new monitoring variables are introduced.

### Table 2: Relationship between Adaptation Triggers and Adaptation Strategies.

<table>
<thead>
<tr>
<th>Change in comput. context</th>
<th>CPU and memory load, incoming/outgoing bandwidth</th>
<th>CPU load &gt; 90%</th>
<th>user migration, replication or instancing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change in business context</td>
<td>number of concurrent users, number of requests per application</td>
<td>number of concurrent users &gt; ∑ user capability of application servers</td>
<td>user migration, replication or instancing</td>
</tr>
<tr>
<td>Prediction values from capacity planning service</td>
<td>number of users/hour, number of requests per application</td>
<td>predicted users &gt; current users + (threshold)</td>
<td>QoS negotiation, replication or instancing/zoning</td>
</tr>
</tbody>
</table>

5 **State of Implementation and Experimental Results**

In this section, we report our initial experimental results with the central part of the edutain@grid architecture, responsible for real-time services for ROIA. We have implemented the Real-Time Framework (RTF) [7] as a C++ based library, since C++ is currently the language of choice for ROIA applications both in industry and
research. Using RTF as a development framework and runtime middleware, we developed several applications, some of them jointly with industrial partners, from three areas: online computer games, e-learning systems, and multi-agent systems for simulating dense crowds. We briefly describe one such application and its use for evaluation purposes.

The RTFDemo application is an industrial-strength, fast-paced online game that takes place in a zoned 3D world and is built on top of the edutain@grid architecture described in Section 3. RTFDemo implements the zoning and the replication adaptation strategies presented in Section 4. The functional and non-functional application requirements of RTFDemo correspond to a typical commercial online game (Table 1), e.g., the game state is updated 25 times per second, both seamless migration between different zones and interactions across zone borders are supported. Each client has control over a single avatar and can move it around and let it interact with the game world and other avatars. Fig 4 shows a screenshot of one client in the RTFDemo game, looking at avatars of other clients.

![Fig 4: Screenshot of RTFDemo (avatars managed by different servers have different colours).](image)

In order to evaluate the zoning and replication adaptation strategies, we conducted experiments that test the capability of different static zoning and replication setups. We use a setup of PCs with 2.66 GHz, CoreDuo 2 CPUs, and 4 GB RAM in a LAN. For experiments, a static setup of zones and servers was started and multiple computer-controlled clients (bots) that continuously sent inputs to their server were connected to the application servers. The average CPU utilization was measured on the application servers as the metric to be evaluated. Clients were allowed to move between zones and thereby generate higher load on some of the application servers.

In the experiments with adaptation by zoning, each zone was assigned to a different server. Figure 5 shows the measured number of players that were able to participate fluently in the game for one, two and four zones, respectively. We observe that the zoning adaptation approach scales almost linearly.

![Figure 5: CPU load for different numbers of zones.](image)
Our second set of experiments aims at the adaptation strategy based on replication: we replicate the computation of a single zone on up to four servers. Figure 6 shows the measured results for the CPU load. One server is able to serve up to 450 clients at a CPU load of 120% (each core has a load below 100%), which is identical to the results of the zoning related tests. But if more servers are added to the processing of a large zone, the client numbers can be increased from the previous limit of 450 clients to up to 1000 clients in a four-server setup. This result shows that the replication adaptation strategy allows games to provide a higher level of interactivity.

**Figure 6: CPU load for different numbers of replicas.**

### 6 Conclusions, Related and Future Work

The main contribution of this paper is the industrial-strength case study in which we applied the generic Lifecycle Model for developing adaptive service-based applications to the novel, emerging class of ROIA (Real-Time Interactive Applications). We demonstrated how the specific, challenging features of ROIA can be met during the evolution and the adaptation cycles of the application development. In particular, we identified the adaptation triggers and adaptation strategies that help to develop high-quality, scalable ROIA applications.

There are previous approaches (e.g., [10], [11]) that deal with the design and realization of service-based applications. Most of them are not flexible since they base the execution of service-based applications on static rules that trigger the execution of a pre-defined adaptation action only when some specific and known events happen. This paper aims at a challenging case when the adaptation needs are dynamic and not all the system characteristics are known a priori, such that the adaptation actions cannot be completely defined at design time. Some other approaches address this issue by proposing an abstraction-based adaptation through which at design time, the adaptation strategies are defined while the concrete mechanism is defined only at runtime. For example, in [12] the design of the application is based on the abstract definition of service. Only at run-time the services are effectively selected on the basis of the situation and context in which the execution is required. The S-Cube Lifecycle Model enables both the built-in and the abstraction adaptation, but it also addresses the dynamic adaptation for which it is possible to provide mechanisms that
select and instantiate adaptation strategies depending on specific triggers and situations [3]. This paper shows the effectiveness of the proposed lifecycle for ROIA applications in which proactive adaptation is mandatory.

Our experimental results confirm the feasibility of applying the S-Cube Lifecycle Model and identified adaptation mechanisms to developing demanding applications. In addition to the RTFDemo described in the paper, we have implemented several industrial applications: a multi-server port of the commercial Quake 3 action game [8], the 3D action game Hunter developed by the video game developer Darkworks and the remote e-learning framework edutain@grid Virtual Classroom [9] developed by the environmental consulting company BMT Cordah Ltd.

Our future work will include implementing the adaptation triggers identified in Section 4, developing design patterns for the described adaptation scenarios, as well as further joint experiments with industrial partners.

References

An Approach to Develop Self-Assembling Self-Adaptive Service Oriented Applications

Luca Cavallaro, Elisabetta Di Nitto, and Matteo Pradella

Abstract—Service-oriented applications are typically built out of existing services possibly made available by third party vendors. The objective is to enable reuse of existing elements (the services), fast development and easy replacement of one service with another, possibly offering better performances or quality of results. Given the fact that adopted services are out of the control of the application, the need for easy replacement could also happen because of the unavailability of a service at runtime. To cope with the above requirements, development of service-oriented application should be supported by proper tools that speed up the identification and selection of services and simplify the work of designers in creating the glue needed to compose services together. Moreover, the application has to be designed to be self-adaptive so that it can autonomously deal at runtime with unavailability of services and changes in the available service-base. In this article we introduce a design approach and a framework to support the construction and operation of self-adaptive service-oriented applications. Our design approach makes possible for a developer to express the requirements of an application in intensive terms and to automatically obtain possible design solutions by running a formal, SAT-based machinery. The identified solutions compensate for the limited information often available at design time by changing some decisions at runtime, when the information becomes available. The runtime framework enables the dynamic replacement of the component services of the application, allowing also the integration of previously unforeseen services. In this way the application can self-adapt reacting to changes in its context or to failures of services.

Index Terms—Service Oriented Architectures, Self-assembling systems, Self-adaptive Applications, Service interface mapping.

1 INTRODUCTION

Service Oriented Architectures (SOAs) are a flexible set of design principles that promote interoperability among loosely coupled services belonging to multiple business domains. Applications are typically composed out of existing services possibly made available by third party vendors. This opens a series of new scenarios that are unimaginable under the closed world assumption [1], which mandates that developers know a priori all the components involved in the system and can model and plan their interactions. On the one hand, a developer would like to exploit as many available services as possible without spending too much time in a direct and comprehensive study of their behavior. On the other hands, he/she would like to make sure that the resulting application is able to replace adopted services, on the fly, in case these last ones become unavailable for any reason or in case new interesting services offering better quality become available.

To cope with the above requirements, development of service-oriented application should be supported by proper tools that speed up the identification and selection of services and simplify the work of designers in creating the glue needed to compose services together. Moreover, the application has to be designed to be self-adaptive so that it can autonomously deal at runtime with unavailability of services and changes in the available service-base.

The literature has proposed in recent some frameworks that enable the automatic synthesis of a service composition (see for instance [2]) and some other frameworks that support design of compositions with reconfiguration in mind (see for instance [3], [4], [5]). However, none of these approaches integrates both issues. Automatic synthesis approaches are often relatively slow in producing possible composition solutions and self-adaptation solutions often require that only reconfigurations foreseen at design time can take place at runtime. This implies that new services becoming available at runtime cannot be considered as replacement of the adopted ones.

In this article we introduce a design approach and a framework to support the construction and operation of self-adaptive service-oriented applications. Our design approach makes possible for a developer to express the requirements of an application intensively, tipically, in terms of the operations that the application is expected to offer and of the ones it requires to be offered by component services. Given this specification, a design solution is automatically obtained by running a formal, SAT-based machinery exploiting the notion of tile-based system. Tile-based systems define computations (in our case, service-based applications) as processes that assemble atomic units called tiles (that represent services). Each tile can be composed only with certain other tiles, according to the symbols they carry. The resulting assembly process is similar to building a jigsaw puzzle with the pieces from a given box (see [6] for an introduction to tile-based systems). The approach shows two interesting characteristics: a) it performs very well by computing service compositions within seconds (see Section 9 for details), b) it can be exploited incrementally, by using parts of solutions computed at one...
stage as the requirement for computing a new solution in a following stage. This implies that, at runtime, it is possible to run the machinery by providing as requirement the current composition minus the pieces that have to be replaced. As a result, other possible solutions are identified, that keep unaltered all it is part of the requirements and replace the rest. Intuitively, the identified solutions compensate for the incorrect or partial information available at design time by changing some decisions at runtime, when the information becomes available.

The ability of using services discovered at runtime and previously unforeseen by the system designer proposes the additional problem of the lack of standardization of their interfaces. In fact newly discovered services may offer interfaces different from those that the composition designer expected to interact with. To cope with this problem we adapt the interface a client expects to the interface that is actually made available by the selected service. The approach is able to deal with conversational services and is based on the synthesis of mapping scripts, finite length sequences of instructions, which translate operations that the client is assuming to invoke on the expected service into the corresponding sequences made available by the service that will be actually used. Such mapping scripts are then interpreted by adapters that intercept all service requests issued by the client and transform them into the requests the services are able to fulfill.

The approach to compute compositions and the one to support interface adaptation have been preliminarily presented in [7] and [8], respectively.

In this article we provide a three levels approach to build self-adaptive service oriented applications that incorporates in a coherent manner the previous two contributions. The paper also presents a software framework that supports the approach and provides an extensive evaluation, showing the practical feasibility of the composition and adaptation models and the efficiency of the framework to run them. This efficiency enables the application to replace at runtime component services, in case of failures, unavailabilities and changes in the application context, recomputing, partially or totally, a new solution when needed.

The rest of the article is structured as follows: Section 2 motivates our work, discussing a realistic case study and presenting some background. Section 3 presents an overview of self-assembling self-adaptive service oriented systems design process using our framework, Section 4 presents the Service-Tiles self-assembly process, Sections 5 and 6 discuss on how services are described, published and then retrieved to constitute the service base that is used as a basis to run the self-assembly process, Section 7 describes the synthesis process for generating mapping scripts, Section 8 presents the framework architecture and its runtime behavior, Section 9 discusses the experimental results we have gathered, Section 10 compares our work with existing approaches in literature and discusses its novelty, finally Section 11 draws some conclusions and a future work plan.

2 MOTIVATIONS AND CASE STUDY

The motivations for our work are the following:

- While service-based applications in theory simplify developers’ work by composing existing pieces of functionality (the services), still, the concrete design and implementation tools that are available force designers to focus on specific and low-level aspects, such as, the syntactic WSDL interface of the services to be used and the XML structure of the data to be exchanged. Moreover, they assume that designers know about information that are not even explicitly stated in the service interface. An example of such implicit information is the protocol needed to converse with these services. Mashup approaches (see [9] or [10] for instance) are trying to address such problems but they do not assume that the composition may have dynamic binding capabilities and make the hypothesis that all the services available for composition are known at design time.

- After development, during execution, services can disappear without the owner of a service composition knowing about it. In the literature approaches to support so called dynamic binding are proposed [5], [3] to address this problem. However, these, in most cases, only enable a service to be replaced by another service offering exactly the same interface. This clearly limits the set of possible substitutions in a context were conceptually similar services are built by different providers and therefore expose different interfaces.

- Again, during execution, those applications that are context-aware could need to interact with different services depending on the context in which they or their users are operating. Dynamic binding is the basic building block that could flexibly support this feature, but it has to be complemented by proper mechanisms that allow the service composition to maintain an explicit concept of context and to select proper services based on such context.

Consider, for example, a traveler assistant application. This supports travelers moving from one city to another using private vehicles and commuting to public transportation when they move in the city. Users are equipped with portable computing devices, which enable them to retrieve information and suggestions about their routes. In case they are moving using private vehicles they exploit car route planning service, while when they are using the public transportation, they rely on information services provided by local transportation authorities. These services offer the possibility to plan user ways through public vehicles and provide other static or realtime information such as time tables, possible problems (e.g., breakdowns or traffic jams), or delays. Part of such information is obtained by gathering and combining data from vehicles moving along the transportation network. Mobility information services can also forecast the status of the transportation network by elaborating sensed weather conditions. For instance, a likely delay of buses can be forecast based on the presence of snow. Thus, when problems are signaled on his/her route, an user may re-plan his/her way.

In order to develop such a system using current techniques, designers should know about the detailed characteristics and
interaces of all involved services. Indeed, designers should build the application so that it explicitly analyzes the location of users (if they are traveling from city to city or if they are staying in a specific city and, in this last case, which one) and, consequently, selects the service that is appropriate to the specific situation. If we aim at building a system that is not peculiar of a specific geographical area, we should not enumerate all possible situations and corresponding services at design time. Instead, we should identify proper mechanisms enabling a runtime, context-dependent selection of services.

Our approach overcomes the problems listed above by postponing some design decisions at runtime, when the needed information is available. This means that the designer, instead of specifying all the application component services and their interactions, should rather specify which requirements are implemented by the application and which requirements should be demanded to third party services, abstracting from the implementation and relying on a self-assembly technique in order to compose them, as they become available at runtime. Table 1 informally lists the requirements, implemented and delegated to others, of the CommuterBuddy application. Intuitively the self-assembly technique we are going to introduce in Section 3 exploits the information about implemented and delegated requirements to compose services, disregarding, at design time, which implementations have to be selected at runtime. The selection criterion mandates that each delegated requirement of the application should be satisfied by a requirement implemented by some available service. The selection is performed according to runtime service availability or to constraints imposed by the application context. If a service becomes unavailable, it is replaced, if possible, by another service. Otherwise, the self-assembly approach is run again to identify a deeper restructuring that still fulfills the delegated requirements. Let’s assume now that CommuterBuddy has been developed to interact with a service called Mobility Information and that, at a certain point, either because of a failure of this service or of a context change, CommuterBuddy has to replace it with Mobility Information2. The two are defined as implementing a solution of the same problem, but they offer a different interface. For instance, a bus stop timetable can be retrieved from Mobility Information by invoking in sequence the operations getRoute, getRouteStops and getStopInfo. The same information can be retrieved from Mobility Information2 by invoking getRouteInfo and getTimetable. As we will discuss in Section 7, if the system execution environment is able to know that the two sequences are equivalent in terms of fulfilled functionality and is able to map the inputs received in the first case into the ones to be received in the second case (and, similarly, the output obtained in the second case into the ones expected in the first case), then, the invocation of Mobility Information2 instead of Mobility Information is still possible.

While usually self-adaptiveness is designed at the level of the component orchestrating all services (CommuterBuddy in our example), there are cases that call for a more decentralized self-adaptiveness. For instance, the effectiveness of mobility information services used by CommuterBuddy may depend on the possibility of retrieving data from local weather services. If weather services manifest unforeseen unavailability or failures, it would be beneficial to have the mobility information services replacing them transparently to the CommuterBuddy service and to its designer.

In the following sections, the CommuterBuddy application will be used as running example to demonstrate our development process and service-composition model. For the purpose of exploiting this example, we assume that our world is populated by the services listed in Table 2. All of them are represented in terms of their implemented requirements, and some of them also expect others to fulfill some requirements. The list features services offering information about vehicles circulating on some roads network, route planning services, weather forecasts services, informational services on the state of mobility in different geographical areas, and, finally, ticket booking and payment gateway services.

### TABLE 1: The CommuterBuddy Application

<table>
<thead>
<tr>
<th>Implemented Requirements</th>
<th>Delegated Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>commuter information</td>
<td>route planning (situation)</td>
</tr>
<tr>
<td></td>
<td>mobility information (location)</td>
</tr>
<tr>
<td></td>
<td>ticket booking</td>
</tr>
</tbody>
</table>

3 Overview of Our Approach

Our approach focuses on three main objectives: i) simplify designers’ work while developing a service-based application by offering a self-assembling mechanism that automatically computes the structure of the application from its requirements, ii) promote decentralization of the self-assembly logic, ii) support self-adaptation of self-assemblies as a reaction to both failures and changes in the execution context. The main hypothesis we make is that the information about available services are stored in a (possibly distributed) registry and that such information concerns the aspects highlighted in the class diagram of Figure 1a. More in detail, each Service in our approach is implemented by one or more ServiceImplementations and it is modeled by a ServiceUnit (this may be related to more than one Service). A ServiceUnit specifies the requirements fulfilled by the corresponding Services in terms of ServicePlugs, and the requirements delegated to other Services in terms of ServiceDocks. Intuitively, the purpose of a self-assembly to find ServicePlugs matching each ServiceDock and to select, among these, one that is actually bound to the corresponding dock. Both plugs and docs have a specification, represented in the figure by the concept PlugDockSpecification. This, in turn, contains an OperationSpec, including information about the signature of each operation and the data it uses as inputs and outputs, and a ProtocolSpec. This defines the order in which operations have to be invoked on a ServiceImplementation. Referring to the example reported in Section 2, Tables 3 and 4 provide the most relevant information contained in an OperationSpec, while Figures 2a and 2b describe the corresponding ProtocolSpecs, respectively. The last relevant element of a ServiceUnit is the CostVector. This contains information about the Quality of Service (QoS) properties associated to the ServiceUnit.
### TABLE 2: Some available services

<table>
<thead>
<tr>
<th>SERVICE NAME</th>
<th>IMPLEMENTED REQUIREMENTS</th>
<th>DELEGATED REQUIREMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Information 1</td>
<td>vehicle information [V1]</td>
<td></td>
</tr>
<tr>
<td>Vehicle Information 2</td>
<td>vehicle information [V2]</td>
<td></td>
</tr>
<tr>
<td>Vehicle Information 3</td>
<td>vehicle information [V3]</td>
<td></td>
</tr>
<tr>
<td>Vehicle Information 4</td>
<td>vehicle information [V4]</td>
<td></td>
</tr>
<tr>
<td>Route planner Milan</td>
<td>route planning [PTMilan]</td>
<td></td>
</tr>
<tr>
<td>Weather forecast Milan</td>
<td>weather forecast [Mi]</td>
<td></td>
</tr>
<tr>
<td>Weather forecast Turin</td>
<td>weather forecast [To]</td>
<td></td>
</tr>
<tr>
<td>Weather forecast North Italy</td>
<td>weather forecast [NItal]</td>
<td></td>
</tr>
<tr>
<td>Weather forecast Rome</td>
<td>weather forecast [Rome]</td>
<td></td>
</tr>
<tr>
<td>Weather forecast Italy</td>
<td>weather forecast [Italy]</td>
<td></td>
</tr>
<tr>
<td>Weather forecast Milan2</td>
<td>weather forecast [Mi]</td>
<td></td>
</tr>
<tr>
<td>Mobility Information Milan</td>
<td>mobility information [Mi]</td>
<td>weather forecast (location) vehicle information (vehicle)</td>
</tr>
<tr>
<td>Mobility Information Turin</td>
<td>mobility information [To]</td>
<td>weather forecast (location) vehicle information (vehicle)</td>
</tr>
<tr>
<td>Mobility Information Rome</td>
<td>mobility information [Rome]</td>
<td>weather forecast (location) vehicle information (vehicle)</td>
</tr>
<tr>
<td>Mobility Information Florence</td>
<td>mobility information [Fi]</td>
<td>weather forecast (location) vehicle information (vehicle)</td>
</tr>
<tr>
<td>Mobility Information Venice</td>
<td>mobility information [Ve]</td>
<td>weather forecast (location) vehicle information (vehicle)</td>
</tr>
<tr>
<td>TicketBooking</td>
<td>ticket booking</td>
<td>Payment</td>
</tr>
<tr>
<td>Payment gateway 1</td>
<td>Payment</td>
<td></td>
</tr>
<tr>
<td>Payment gateway 2</td>
<td>Payment</td>
<td></td>
</tr>
</tbody>
</table>

(a) Main elements of a service specification.

(b) Three levels composition model.

Fig. 1: Conceptual model of our framework
### TABLE 3: Operations, Input and Output parameters for the Mobility Information Rome service

<table>
<thead>
<tr>
<th>OPERATION NAME</th>
<th>INPUT PARAMETERS</th>
<th>OUTPUT PARAMETERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>getRoute</td>
<td>start, end</td>
<td>route</td>
</tr>
<tr>
<td>getRouteStops</td>
<td>route</td>
<td>stops</td>
</tr>
<tr>
<td>getStopInfo</td>
<td>stop</td>
<td>time</td>
</tr>
<tr>
<td>getTrafficInfo</td>
<td>stop</td>
<td>traffic</td>
</tr>
</tbody>
</table>

### TABLE 4: Operations, Input and Output parameters for the Mobility Information Milan service

<table>
<thead>
<tr>
<th>OPERATION NAME</th>
<th>INPUT PARAMETERS</th>
<th>OUTPUT PARAMETERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>getStops</td>
<td>startLat, startLong, endLat, endLong</td>
<td>route, stops</td>
</tr>
<tr>
<td>getTimeInfo</td>
<td>route, stop</td>
<td>time</td>
</tr>
<tr>
<td>getTrafficInfo</td>
<td>route, stop</td>
<td>traffic</td>
</tr>
</tbody>
</table>
Our approach works at three levels of abstraction (see Figure 1b). At the specification level, we assume that services are specified and published. At this same level the designer of a service composition specifies the requirements he/she wants the system to fulfill in terms of one or more DockSpecifications. Given such specifications, the registry is searched for identifying matching PlugSpecifications. Starting from these last ones, an assembly is built at the composition level, where docks are actually bound to plugs also depending on context and cost information. Such a process is automated and is able to address also the cases where the bound services have themselves specific requirements to be fulfilled by other services. At the communication level, docks and plugs are checked to see if they actually match from the syntactic point of view. If not, proper mapping scripts are automatically generated, starting from the knowledge of the operation and protocol specification. Assemblies and mapping scripts can be recreated/modified at runtime if this is needed to cope with context changes and/or failures/unavailabilities of components.

In the detailed presentation of the three levels, we start from the intermediate one (Section 4 that is where the Service-Tiles machinery takes place, then we move to the specification level (Sections 5 and 6) and finally to the communication level (Section 7).

4 Automatic Creation of Assemblies

The composition level allows the designer to find consistent compositions of service units from a given service base. The rules of our composition model definition must ensure that services are composed correctly. While the notion of correct service composition is a very broad one, here we focus on a specific — yet significant — aspect, namely the problem of creating a composition by selecting service units in such a way that every dock is satisfied by some plug offered by another unit in the composition. In other words, the solution to our problem is a composition that “works” when considered in its entirety.

4.1 Self-Assembly using Service-Tiles

In order to compose a “working” application the plugs and docks defined in the specification level and grouped in service units are used to self-assemble a service composition by selecting some available service units in such a way that it is possible to find a plug matching each dock of the application. Intuitively, a service unit in a compound can offer the services associated to its plugs in O only if it works properly, that is only if all requirement demanded to external services, represented by docks in R, are satisfied by some matching plugs belonging to another chosen unit. Correspondingly, the service system building problem requires to compose service units from a service base in such a way that they can function properly, that is all requests of all instantiated units are satisfied.

Definition 1 (Service-building problem). Given: a) a service base \( S = (\Sigma, k, \mathcal{U}) \); b) a multi-set \( I : \mathcal{U} \rightarrow \mathbb{N} \) of initial service units; c) a cost bound \( K = [K_1, \ldots, K_k] \in (\mathbb{N} \cup \{\infty\})^k \); determine if there exists a multi-set \( T : \mathcal{U} \rightarrow \mathbb{N} \) of service units from the base such that: i) \( I \subseteq T \). ii) Every dock from a service unit in \( T \) is satisfied by the offered plug of some service unit also in \( T \). This is formalized by requiring that \( \mathcal{R} \subseteq \mathcal{O} \), where, \( \mathcal{R} : \Sigma \rightarrow \mathbb{N} \) is the multi-set of docks defined by \( \mathcal{R}(x) = \sum_{x \in U \mid O} T(U) \) and \( \mathcal{O} : \Sigma \rightarrow \mathbb{N} \) is the multi-set of plugs defined by \( \mathcal{O}(x) = \sum_{x \in U \mid O} T(U) \), where \( U \in \mathcal{U} \).

Notice that in our formal model any dock of any unit can be connected with the corresponding plug of any other unit in the solution. Hence, a notion of bi-dimensional locality — usually present in traditional tile-based systems — is lost, because the requirements for a correct composition are ultimately just on the cardinality of plugs and docks and not on their spatial displacement.

Intuitively, \( I \) represents the input that the system designer provides to the the problem. This input usually consists in one or more service units that usually represent the part of the application that the designer builds himself. In the running example of Section 2, \( I \) would represent a single instance of the Commuter Buddy application, for which the designer wants to find a suitable binding.

\( \mathcal{U} \) represents the service base including all service units that are available at the instant in which the solution for the service-building problem is computed. Figure 5 pictures such a service base referred to the case study in Section 2.

Given \( I \) and \( \mathcal{U} \), the service-building problem consists in finding an assembly of service units from \( \mathcal{U} \) which satisfies all the requests of the service units in \( I \) and do not exceed the bounds on costs. For instance, we can represent as a tile of our model Ticket Booking = [+ Ticket booking, – Payment] and we can assume it as \( I \). Moreover we can associate a cost bound of 4 to the Ticket Booking unit. In order to solve the service-building problem for this initial set we need to find a service unit offering the Payment plug and having a maximum cost of 4. Considering as \( \mathcal{U} \) the service base presented in Figure 5, the unit PaymentGW1 = [+ Payment] can be composed in such a way that all requests are satisfied, if we make the hypothesis that using the Payment plug has a cost of 4.

The service-building problem can be used both for determining a binding for the application when it is designed and to adapt the application determining a rebinding at runtime, in case the component services, or part of them, become unavailable or fail at runtime. In fact it is possible to partially recalculate the application binding defining an initial set \( I \) comprising those services in the assembly that should not be substituted and deleting from \( \mathcal{U} \) those failed services that need to be substituted.

1. Multi-sets are sets with repeated elements. With standard notation, a multi-set \( A \) over elements in \( D \) is represented by a function \( A : D \rightarrow \mathbb{N} \) such that \( A(d) \) is the cardinality of element \( d \in D \) in \( A \). The notion of subset is extended to multi-sets as customary: \( A \subseteq B \) for multi-sets \( A, B : D \rightarrow \mathbb{N} \) if \( A(d) \leq B(d) \) for all \( d \in D \).
2. \( A|_{/B} \) denotes the projection of tuple \( A \) over set \( B \), i.e., \( B \)'s component in \( A \).
4.2 Encoding the service-building problem with integer linear programming

Here we show how to solve instances of the service-building problem (Definition 1) by encoding them as an Integer Linear Programming (ILP) problem. ILP problems consist in minimizing the value of a linear function of some integer-valued variables subject to a set of linear inequality constraints on the same variables [11]. ILP is a natural formulation for many optimization problems. The technology for ILP solving is very mature: even if ILP is an NP-complete problem, a variety of practically very efficient off-the-shelf ILP solvers are available.

Given that the service-building problem is NP-complete (see the proof in [7]), this is the best possible encoding from a worst-case complexity viewpoint.

A service-building problem for generic service base $S = (\Sigma, k, \mathcal{U})$, initialization $i$, and cost bound $K$ can be solved by encoding it as an ILP problem of polynomial size in $|\mathcal{U}| + |\Sigma| + k$, $\max_x I(x)$, $\max_{1 \leq i \leq k} K_i$. To this end, consider $|\mathcal{U}|$ integer variables $u_1, u_2, \ldots, u_{|\mathcal{U}|}$, and let $\Sigma = \{x_1, x_2, \ldots, x_{|\Sigma|}\}$. Let us introduce the binary-valued functions $R(x_k, u_i), O(x_k, u_i)$ with $x_k \in \Sigma$ defined as the indicator functions $1_{U_i \in R}(x_k)$ and $1_{U_i \in O}(x_k)$ of the sets $U_i \in R$ and $U_i \in O$, respectively. $C_i = [c_{i1}, c_{i2}, \ldots, c_{ik}]$ denotes the cost vector of the service unit corresponding to $u_i$. The system of $|\mathcal{U}| + |\Sigma| + k$ linear inequality constraints and one objective function reported in formula (1) encodes the service-building problem.

The first $|\mathcal{U}|$ inequalities encode the data about the initial service units $I$: the following $|\Sigma|$ inequalities encode the requirement that the total amount of offers for any service must be no fewer than the total amount of requests for the same service; the last $k$ inequalities constrain the total costs not to exceed the values in $K$.

A solution of the ILP problem of formula (1) is a vector of nonnegative integer values, one for each service unit in $\mathcal{U}$. Each value in the vector specifies how many instances of that service unit are included in the composition. Correspondingly, it is straightforward to compute the the total cost of the composition.

$$\begin{align*}
\min & \sum_{1 \leq i \leq |\mathcal{U}|} \left( \sum_{1 \leq j \leq k} c_{ij} \right) u_i \\
\text{s.t.} & \quad u_1 \geq I(u_1) \\
& \quad \vdots \\
& \quad u_{|\mathcal{U}|} \geq I(u_{|\mathcal{U}|}) \\
& \sum_{1 \leq i \leq |\mathcal{U}|} (O(x_1, u_i) - R(x_1, u_i)) u_i \geq 0 \\
& \quad \vdots \\
& \sum_{1 \leq i \leq |\mathcal{U}|} (O(x_{|\Sigma|}, u_i) - R(x_{|\Sigma|}, u_i)) u_i \geq 0 \\
& \sum_{1 \leq i \leq |\mathcal{U}|} -c_{i1} u_i \geq -(K_1 + 1) \\
& \quad \vdots \\
& \sum_{1 \leq i \leq |\mathcal{U}|} -c_{ik} u_i \geq -(K_k + 1)
\end{align*}$$

Table 5 shows a possible solution to the ILP model of the service-building problem for the example introduced in Section 2. The solution is a “snapshot” of a possible valid assembly under the assumption that the current location is $[Mi]$ (Milan) and where a mono-dimensional cost representing the response time of services is considered. The Cardinality column represents how many instances of a service unit of a certain type are needed in the solution.

4.3 Context representation in the service-building problem

Let us point out how take into account context information in the service-building problem. The matching between plugs and docks is performed on a syntactical base in the service

---

Fig. 2: Protocol LTSs for Mobility Information Rome (2a) and Mobility Information Milan (2b)
TABLE 5: An example of solution for the service-building problem formulated as ILP.

<table>
<thead>
<tr>
<th>Service Unit</th>
<th>Cardinality</th>
<th>Response Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Information 1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Vehicle Information 2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Vehicle Information 3</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Vehicle Information 4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Route planner</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>WeatherForecast Milan</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>WeatherForecast Milan 2</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>WeatherForecast Turin</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>WeatherForecast North Italy</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>WeatherForecast Rome</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>WeatherForecast Italy</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Mobility Information Milan</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Mobility Information Turin</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Mobility Information Rome</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Mobility Information Florence</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Mobility Information Venice</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>TicketBooking</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Payment gateway 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Payment gateway 2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Total Cost (response time)</td>
<td></td>
<td>9</td>
</tr>
</tbody>
</table>

composition level. Consequently the parameters for docks and the constraints for plugs introduce a problem. Consider for instance the mobility information (location) unit Commuter Buddy in the case study. As shown in Figure 5 there is no plug in the service base marked with the same label.

To solve this problem we augment the set \( U \) of available service units in the service-building problem, with a few context units. These do not represent any “real” service unit but have the only purpose of modeling the context. Each context unit offers a plug marked with the same label of some dock required by one of the service units in \( U \) and requires some docks which are used to select the service unit which is appropriate in the current context.

Consider for instance the example reported in Figure 3. The context units Mobility Information (location) = [Mi] and Weather forecast (location) = [Mi] respectively represent the context dependences of dock mobility information (location) of unit Commuter Buddy and weather forecast (location) of unit Mobility Information Milan. For instance, the context unit Mobility Information (location) = [Mi] offers the plug Mobility Information (location) and requires the dock Mobility Information [Mi]. This context unit is introduced in the service base when the user location is [Mi] and guarantees the inclusion of unit Mobility Information Milan. The context unit Weather forecast (location) = [Mi] serves a similar role for weather forecast.

5 PUBLICATION OF SERVICES

The publication process consists in the service publisher developing services and publishing them in the registry, in order to make them accessible by third parties composition designers. The interactions of the two roles happen through a registry which organizes service implementations associating each of them to a plug. Each plug is related to an abstract operations specification that details the properties of the data exchanged by the service operations as input and output parameters.

![Fig. 3: An example of context units to represent context for Mobility Information and Weather Forecast](image)

When a publisher wants to make his service available for third parties he has to decide if he has to associate it to an existing plug in the registry, or if he needs to create a new plug. In order to take this decision the designer retrieves some of the available plugs querying the registry and analyzes their abstract operations specifications. This analysis consists in trying to verify if some desired properties hold on the retrieved specification. In case a description verifying the desired properties is found the service implementation is stored in the registry and associated to the label referred to the description. In this case the service will be modeled in the composition level as a service unit offering a plug marked with that label. In case no suitable description is found in the registry, the developer should write himself a description for his service and publish it in the registry as a new plug.

Let us clarify the publication process with an example based on the case study presented in Section 2. In this case we assume that a service publisher has just developed a service to provide mobility information about the city of Milan. The publisher queries the registry in order to discover what other services implementing that requirement exist. In our example the publisher does not find any suitable service for his query, so he decides to create a new plug in the registry. To do so he needs to build the whole service interface specification, comprising the abstract operations specification, the operation specification and the protocol specification.

The abstract operations specification predicates about the data that are exchanged during the interaction with the external service and imposes some constraints on them. This description can be expressed using any logic, however in this article we are going to use Alloy [12] as a reference, since it provides a stable tool for reasoning on such descriptions. A mobility information description example is reported in Listing 1,
where a subset of the Alloy model describing how to obtain public transportation information using the service is defined. The model is composed of some signatures, which model the type of data that will be exchanged interacting with an invoker, defines some facts, which represent some constraints on the modeled data, and eventually defines some predicates, which show how the signatures should be used to retrieve the desired information from the service. In the example the data exchanged by the service carry information about the user location, his route, and define the public transportation map, including the transportation lines and their stops. The fact reported in the example, called contextSensitiveMap, remarks the context sensitivity of the Mobility Information services. This fact constrains a public transportation map stating that user requests regarding the public transportation should only concern data contained in a public transportation map. The example also features the mobilityInfo predicate, used to model at an abstract level the operation that the service is going to offer. The predicate shows a model of the system corresponding to a service request to retrieve mobility information: the invoker specifies an userRoute, a ptMap representing the public transportation map for a city. The userRoute should point out the user location and should contain some Stops, such as they are also contained in the ptMap. Given these inputs the service returns some Stop such as they are near to the user location and have some public transportation lines connecting them.

```
some sig Location{}
some sig timetable{
    hour: one Int,
    minute: one Int,
}
one sig userRoute{
    start: one Location,
    end: one Location,
    startTime: lone timetable,
    endTime: lone timetable,
    referenceMap: one ptMap
}
sig ptMap{
    transportations: set ptLine,
    places: set Location
}
some sig ptLine{
    route: set Stop
}
some sig Stop{
    loc: one Location,
    time: set timetable,
    lines: set ptLine
}

fact contextSensitiveMap{
    all ust: userRoute.start,
    uend: userRoute.end, umap: userRoute.referenceMap|
    ust ≠ uend
    and ust in umap. transportations.route.loc
    and uend in umap. transportations.route.loc
}
pred mobilityInfo[ userRoute: userRoute, map: ptMap]{
    some line:ptLine, lend, lstart: Stop,
    ustart:usroute.start, uend:usroute.end | line in ptMap.transportations
    and lend in line.route
    and lstart in line.route
    and lstart in (loc>=ustart).ustart
    and lend in (loc>=uend).uend
    and usroutestartTime in lstart.time
    or uroute.endTime in lend.time
}
```

Listing 1: An example of alloy specification for mobility information plug

6 Service retrieval and construction of a service base

A service composition designer in the specification level has to individuate the requirements that his application will demand to external services. Those requirements are identified in the application under development by docks. The composition designer determines the application docks retrieving some plugs from the registry, examining their associated specifications and, in case a specification verifies the properties the designer states for the dock under design, using the label that identifies the plug to mark the dock. The service implementations associated to the plugs individuated by the composition designer will contribute to the service base building process. The service base will be used in the composition level to determine an assembly of services.

6.1 Retrieving services from abstract operations specification

The process starts when a designer queries the registry to retrieve some of the available abstract operations description. The retrieved descriptions are examined by the designer which defines some desired properties on them and tries to verify if such properties hold. In case one of the retrieved descriptions verifies the desired properties the associated plug is used to mark the dock under design.

Figure 4 illustrates the service retrieval process using the case study in Section 2. A designer developing the Commuter-Buddy application foresees the need to demand to an external service the retrieval of information about the mobility in a certain city. This means that he is going to search for a service providing information about the public transportation table and about the traffic in a certain city area. The service that is going to fulfill this requirement will consequently be context sensitive (i.e., will change according to the city the application user is in). The developer queries the registry and retrieves the specification shown in Listing 1. In order to prove the desired properties on the retrieved specification the developer writes the assertion called rightInfo, shown in Listing 2. This
assert rightInfo{
    some uroute: userRoute, map: ptMap, s1, s2: Stop |
    getInfo(uroute, map) implies
    (s1.lines in map.transportations
    and s2.lines in map.transportations
    and ![s1.lines & s2.lines] ≠ 0
    and uroute.start = (s1.loc > uroute.start)
    and uroute.start = (s2.loc > uroute.start))
}

Listing 2: A predicate and an assertion concerning service description in Listing 1

Fig. 4: Details of the specification level introduced in Figure 1b, instantiated on the case study in Section 2.

assertion is concerned with the consistency of information retrieved form the Mobility Information service and states that when some mobility information is requested then some stops should be retrieved such as they are contained in the public transportation map for the city where the user is, and are situated near the location specified by the user.


to prove his predicate and assertions for a sufficient bound the developer can be convinced that the dock associated to the retrieved description can fit his requirements, or, in case some inconsistency is found, can continue browsing the registry. In case no service satisfying the developer needs is found, he should chose whether changing the requirements or deciding to implement himself the requirement. In our case study the designer specifies a bound of 5 for the mobilityInfo predicate and a bound of 15 for the rightInfo assertion. For the assertion in Listing 2 no counterexample is found, within a bound of 15. This means that the property holds when in the model there are up to 15 instances of each signature. When the developer is convinced that the retrieved plug description matches his expectation he associates a dock, having the same identifier of the selected plug, to the application under development.

After going through the described process for the example the developer finds that the description associated to the symbol mobility information verifies his desired properties, consequently this symbol will mark the dock under development. The composition designer can decide to capture also the context dependence of the designed dock, appending it a parameter. Considering the mobility information dock, the composition designer can consider its dependence from the user location, so he will mark the dock with the mobility information (location) symbol.

In the composition level those service implementations offering some plugs matching the designed dock will be represented as service units offering that plug. Referring again to Figure 4, Service Implementation 1, Service Implementation 2 and Service Implementation 3 will be represented in the service building problem as service units offering a plug marked with the label mobility information. It is interesting to notice a service implementations can be able to provide information for a specific only. Considering the service implementations in Figure 4, for instance Mobility Information Milan can provide information for Milan only. In order to represent this restriction service implementations are marked by the service publisher using a constraint. When the service implementation will be represented in the service base as a service unit it will offer a plug marked with mobility information [Mi], where the [Mi] constraint indicates that the service provides information for the city of Milan, instead of mobility information.

6.2 Building the service base

Once the docks of the application under development have been defined by the composition designer it is necessary to build the service base that will be used as input for the composition level. The composition level deals with how to compose together into a working application some services from a finite set. In our approach the composition is done relying on a new kind of tile-based self-assembly technique, called Service-Tiles, that we introduced in [7]. The service base is built retrieving from the registry all those service implementations that can be represented by a service unit offering a plug matching one of the docks of the application under development. It is interesting to notice that those service units that are included in the service base may require
themselves to demand some requirements to external services. Consider, for instance, the Mobility Information Milan service and its representation in terms of implemented and external requirements in Table 2. That service requires to demand to external services the collection of information about the traffic and about vehicles. This implies that the Mobility Information Milan service should be represented in our model as a service unit offering a mobility information [Mi] plug and requiring a weather forecast and a vehicle docks. Consequently some service units offering some plugs matching those docks have to be included too in the service base. The building process iteratively augments the service base adding all those service units offering those plugs required by some unit already in the service base, until there are no more docks not matched by any plug. An example of service base for the case study in Section 2 is reported in Figure 5.

In the composition level we represent plugs and dock syntactically, with symbols from a finite alphabet $\Sigma$, grouped into service units, which model service implementations and are comparable to tiles of a tiling system. For instance, mobility information and ticket booking are symbols of the alphabet $\Sigma$ in the running example (see Table 2). Each unit comes with a vector of (scalar) costs of arbitrary size (possibly empty). An element in the cost vector represents the magnitude of a certain cost dimension resulting from using the plugs offered by the service unit.

The $\Sigma$ alphabet is in part built by the service composition designer in the dock definition process, when he associates the labels found in the service registry to the docks of the application under development, and in part by the service publishers. In fact on one hand service publishers can define new plugs in the registry, on the other hand the service implementations they publish can themselves demand some requirements to external services.

Formally we can define a service unit and a service base as follows.

**Definition 2.** A service unit $U$ is a tuple $\langle \Sigma, C, O, R \rangle$ of plugs $O = \{o_1, o_2, \ldots\}$ and docks $R = \{r_1, r_2, \ldots\}$ labeled with symbols from alphabet $\Sigma$, together with a cost vector $C = [c_1, c_2, \ldots] \in \mathbb{N}^{|C|}$.

A service unit $U$ will be conveniently represented as $\langle \{c_1, o_2, \ldots : +o_1, +o_2, \ldots, -r_1, -r_2, \ldots\} \rangle$. A collection of service units over a common alphabet with cost vectors of the same size constitutes a service base.

**Definition 3.** A service base $S$ is a tuple $\langle \Sigma, k, \mathcal{U} \rangle$, where $\mathcal{U} \subseteq \mathbb{N}^k \times 2^\Sigma \times 2^\Sigma$ is a set of service units over symbols in the common alphabet $\Sigma$, each with its cost vector of size $k$.

The service base so defined is used to automatically compute an assembly, providing for each dock defined by the service composition designer one service unit offering a plug matching it. This problem is formulated as the service-building problem and is formalized and explained in the next section.

### 7 Creation of Communication Adapters

Service-Tiles allow the integration of services unforeseen by the designer during the application runtime. Using this technique, service units offering a plug, identified by a certain label, can be can be bound to other service units requiring a dock identified with the same label. When necessary a service unit bound to an application can be substituted with another service unit offering the same plug. This feature of Service-Tiles introduces a new problem, since service interfaces lack standardization. In fact even if plugs and docks are composed together on the base of the label they are identified by, there is no guarantee that the service interface specification associated with a dock is the same of the one that the selected plug offers. Consider for instance the examples in Figure 2. Both the services whose protocol is represented can be associated to the mobility information plug, although they provide an implementation of that requirement for two different cities.

4. More explicitly, each element $(C, O, R) \in \mathbb{N}^k \times 2^\Sigma \times 2^\Sigma$ represents a service unit $\langle \Sigma, C, O, R \rangle$. 
This implies that when an user is in Rome he should use Mobility Information Rome, when he is in Milan he should use Mobility Information Milan. Since the two services show differences both in their operations descriptions and in their protocols if Commuter Buddy were to design to work with Mobility Information Rome it could not be able to exploit Mobility Information Milan. Let us consider for instance the sequence of operations needed to retrieve a time table for a public transportation stop in Mobility Information Rome. An user wanting to retrieve the timetable for a given stop should invoke the following operations sequence: getRoute, getRouteStops, getStopInfo. If a Commuter Buddy user moves to Milan he should invoke the following sequence to retrieve the same result: getStops, getTimeInfo. Moreover the two services have some differences also between their operations specifications (see Tables 3 and 4).

A service designer willing to integrate both services in his/her application can write by hand the necessary code to make the application tolerate the protocol and operation mismatches (see [14] for a classification of mismatches). In that case it would be necessary to code by hand also the code for integrating mobility information services for other cities (e.g. Mobility Information Florence or Mobility Information Venice). This would limit the services that can be bound to the application just to those that a designer can foresee at design time, hampering the benefits of Service-Tiles.

### 7.1 Substitutable sequences

To solve both operations and protocol mismatches, the interface level introduced in Figure 1b exploits a methodology to enable service substitution through the automatic definition of suitable mapping scripts (see in [8] and [15]).

Mapping scripts are automatically derived given the service interface specifications respectively expected by the invoker and provided by the invoked service. The mapping between an expected and an actual service assumes that the compatibility between data has been previously defined. This relation allows us to map pieces of data of different services on the same label, solving parameter mismatches. Literature proposes many techniques to establish data compatibility and in [16] we proposed ourselves a semantics based technique that allows us to map different data, representing the same concept, on a label. Since service classification in this article is performed on the base of an abstract operations specification, which introduces the data exchanged during the service interaction, we assume that two pieces of data belonging to different services operations are “compatible” if they can be mapped on the same label taken from the set of Alloy signatures or they can be mapped on the same relation taken from one signature of the specification associated to the selected plug. An example of this relation for the example in Figure 2 is shown in Table 6.

Given this compatibility definition, our notion of substitutability mandates that two operation sequences, $seq_{exp} = o_{1}^{exp}, o_{2}^{exp}, \ldots, o_{g}^{exp}$ on the automaton of the expected service and $seq_{act} = o_{1}^{act}, o_{2}^{act}, \ldots, o_{h}^{act}$ in the automaton of the actual service, are substitutable if a client designed to use the expected service sequence can use the actual service sequence without noticing the difference. This can happen when the actual sequence operations require as input at most all of the data provided as input to the expected sequence operations and return at least all the data the expected sequence is expected to return to the invoker.

The rationale of this definition can be understood thinking of the service substitution process. In this process a client is designed to interact with an expected service and, therefore, expects to invoke the expected service operations, providing for each invocation the input parameters required by the operation and expecting as output the output parameters provided by the operation. When the expected service is substituted by an actual service, in order for the client to be unaware of the change, an actual service operation should be able to work with the data the client expects to provide and should provide the data the client is expecting to be returned by the invocation.

Formally, we consider two sets $D = \bigcup_{1 \leq i \leq n} D_{i}$ and $E = \bigcup_{1 \leq j \leq m} E_{j}$, respectively representing all the data types that can be used as input and output parameters in the sequence. For each operation $o_{exp}$ in the expected service operations description and for each operation $o_{act}$ in the actual service operations description we define two sets of functions $In_{o_{exp}}: D \rightarrow \mathbb{N}$, and $Out_{o_{act}}: E \rightarrow \mathbb{N}$. These functions map each of the data types that can be used as input, respectively as output, to a natural number, describing how many instances of that type the operation takes as input parameter, respectively returns as output parameter. Consider for instance the operation getRoute in the expected sequence. $In_{getRoute}(userRoute.start) = 1$, $In_{getRoute}(userRoute.end) = 1$, and $Out_{getRoute}(ptLine) = 1$.

We also consider the $O_{exp}$ and the $O_{act}$ sets, which respectively contains all the expected and actual service operations. For instance, assuming the services whose operation specifications are reported respectively in Table 3 and Table 4 as expected and actual service, the $O_{exp}$ set is composed of the operations shown in column operation name of Table 3, and the $O_{act}$ is composed of the operations shown in column operation name of Table 4.

Given the definitions of the aforementioned sets and functions we can define a parallel composition of the expected and actual sequences as one of the possible $\mathcal{R} = seq_{exp} \parallel seq_{act}$, where $\mathcal{R} = R_{1}, R_{2}, \ldots, R_{g}, \ldots, R_{h}$. For each $1 \leq g \leq h$ each pair $R_{g} = (r_{1}, r_{2})$, is such that:

$$r_{1} = R_{g}|r_{1} \in O_{exp} \cup \{\epsilon\}$$

$$r_{2} = R_{g}|r_{2} \in O_{act} \cup \{\epsilon\}.$$
We can call $R_1$ (respectively $R_2$) the sequence obtained by projecting $R$ on the first component (respectively on the second component). Finally we can define the homomorphism $\not\in$ that deletes all the instances of $\epsilon$ in the projections $R_1$ and in $R_2$, such that:

\[
\not\in (R_1) = seq_{exp} \\
\not\in (R_2) = seq_{act}.
\]

Considering again getRoute, getRouteStops, getStopInfo as $seq_{exp}$ and getSteps, getTimeInfo as $seq_{act}$ a possible instantiation of $R$ could be $(getRoute, \epsilon), (\epsilon, getSteps), (getRouteStops, \epsilon), (getStopInfo, \epsilon), (\epsilon, getTimeInfo)$. It is interesting to notice that given a $seq_{exp}$ and a $seq_{act}$ their parallel composition does not result in an unique $R$ instance. Consequently the one we are considering here is just a possible instance of $R$. Moreover, even if in this case for each $1 \leq g \leq h$, $R_g|_1$ or $R_g|_2$ is $\epsilon$, the definition of $R$ does not mandate that this condition is always verified.

In order to find a sequence $seq_{act}$, substitutable to a sequence $seq_{act}$ we need to keep track of the data exchanged between the client and the expected service introducing two sets of functions:

- $\text{seen}(R_{g-1}, D_i) + \text{In}_{R_{g1}}(D_i) - \text{In}_{R_{g2}}(D_i)$ with $\text{seen}(R_g, D_i) \in \mathbb{N}$ and $\text{seen}(R_0, D_i) = 0$.
- $\text{needed}(R_{g-1}, E_j) + \text{Out}_{R_{g1}}(E_j) - \text{Out}_{R_{g2}}(E_j)$ with $\text{needed}(R_g, E_j) \in \mathbb{Z}$ and $\text{needed}(R_0, E_j) = 0$.

Intuitively the $\text{seen}(R_g)$ and the $\text{needed}(R_g)$ functions keep track of how many instances of a type of input, respectively output, parameters are available in one step of $R$.

The $\text{seen}(R_g, D_i)$ function is used to define a progress property on the $R$ sequence; the $i^{th}$ step of the sequence can be performed only if after his invocation $\text{seen}(R_g, D_i)$ $\geq 0$ for each $D_i$ in $D$. Intuitively, when an actual sequence operation is substituted to an expected sequences operation the client sends an invocation that are directed to the latter. The invocation contains those input parameters that the expected service operation requires as input. In order for the client to be unaware of the substitution the operation of the actual sequence invoked instead of the expected one, should work with at most the input parameters that the client expects to send.

The $\text{needed}(R_g, E_j)$ is used to define a consistency property on the $R$ sequence: after the sequence reaches its last step, for each $E_j$ in $E$, $\text{needed}(R_g, E_j) \leq 0$. Intuitively, when the client performs an invocation for an abstract sequence operation it expects the service to return the set of output parameters declared in the operation description. In order for an actual sequence operation to substitute the expected one, without the client being aware of it, it is necessary for the actual operation to return at least all the parameters that the client expects.

Given the above definitions we can define our notion of substitutability.

**Definition 4** (substitutability). A sequence of operations $seq_{exp}$ on the expected service automaton, is substitutable with a sequence $seq_{act}$ on the actual service automon if given a possible $R = seq_{exp} \parallel seq_{act}$ the progress property holds for each step of $R$ and the consistency property eventually holds from a given step of $R$ on.

It is noticeable that our substitutability definition covers the most general case of interaction between a client and a service, in which the invocation semantics is *asynchronous*. This means that the invocation is non-blocking, therefore the client can proceed with his workflow and possibly invoke other service operations without waiting for the response to the first invocation. The service will eventually send his response to the client when that will be available. The case in which the invocation semantics is synchronous (i.e. the client waits for the service reply before proceeding) falls under our definition, considering a more restrictive notion of the progress condition in which, once a pair $R_g$ having a $R_g|_1 = \sigma_g^{exp} \neq \epsilon$, figures on the current step of $R$, and $\text{seen}(R_g, E_j) > 0$ for some $E_j$ in $E$, then the following steps of $R$ will be $\epsilon$, $R_g|_2$, until a step $R_{g+c}$ of $R$ such as $\text{seen}(R_{g+c}, E_j) = 0$.

Considering again the example of $R$ example given in this section, the first step of the sequence $R_1$, is composed of the $(getRoute, \epsilon)$ pair. The $\text{seen}(R_1, D_i)$ functions assume the following values: $\text{seen}(getRoute, \epsilon).userRoute.start) = 1$, $\text{seen}(getRoute, \epsilon).userRoute.end) = 1$. This means that the operation getRoute is performed on the expected sequence, while no operation is performed on the actual sequence. The getRoute operation takes as input the userRoute.start and userRoute.end parameters, while no input parameters is associated with the $\epsilon$ operation. The $\text{needed}(R_1, E_j)$ functions assumes the following values: $\text{needed}(getRoute, \epsilon, ptLine) = 1$. This means that the client expects the operation getRoute to return the ptLine parameter.

The next step $R_2$ of the sequence can be performed in case the progress property is verified, which for this example means that the actual operation sequence an operation taking as input at most one instance of userRoute.start and one instance of userRoute.end parameter can be invoked. In the example $R$ sequence the $R_2$ step is a pair of $(\epsilon, getSteps)$. The getSteps operation requires as input an instance of userRoute.start and an instance of userRoute.end. This meets the profess property and, after the invocation of the operation, the $\text{seen}(R_2, D_i)$ functions have the following values: $\text{seen}(getRoute, \epsilon).userRoute.start) = 0$, $\text{seen}(getRoute, \epsilon).userRoute.end) = 0$. The getSteps actual sequence operation also returns an instance of ptLine and an instance of ptLine.route. Consequently the $\text{needed}(R_2, E_j)$ functions show these values: $\text{needed}(\epsilon, getSteps), ptLine) = 0$, $\text{needed}(\epsilon, getSteps), ptLine.route) = -1$. The latter value means that the ptLine.route instance returned is not currently needed as an expected service response and will be considered for later use.

The step $R_3$ of the sequence is composed of the pair $(getRouteStops, \epsilon)$. In this case also the progress property is verified since $\text{seen}(getRouteStops, \epsilon).ptLine) = 1$ after the invocation of this sequence step. The $\text{needed}(R_3, E_j)$ function takes the values: $\text{needed}(getRouteStops, \epsilon), ptLine) = 0$, since the getRouteStops operation requires as output parameter an instance of ptLine.route.
The $R_4$ step is the pair $(\text{getStopInfo}, \epsilon)$. It is interesting to notice that this step presents an operation invoked again only on the expected sequence and not on the actual sequence. This happens since for all data type $D_i$ in $D$ the value of $\text{seen}(R_3, D_i) = 0$. Consequently invoking any actual sequence would make some $\text{seen}(R_4, D_i)$ function run to a value less than 0. This violates the progress condition. After the $R_4$ step $\text{seen}((\text{getStopInfo}, \epsilon), \text{Stop}) = 1$ and $\text{needed}((\text{getStopInfo}, \epsilon), \text{timetable}) = 1$.

Step $R_5$ contains the pair $(\epsilon, \text{getTimeInfo})$, which makes the $\text{seen}(\epsilon, \text{getTimeInfo}), \text{Stop}) = 0$ and the $\text{needed}(\epsilon, \text{getTimeInfo}), \text{timetable}) = 0$. This meets the consistency condition, which makes $\text{seq}_{\text{exp}}$ in the example substitutable by $\text{seq}_{\text{act}}$.

### 7.2 Generating mapping script for substitutable sequences

In order to generate mapping scripts we built a reasoning mechanism that given some $\text{seq}_{\text{exp}}$ returns a substitutable sequence of operations $\text{seq}_{\text{act}}$. The reasoning mechanism has been formulated using the linear temporal logic language described in [15]. The mapping scripts generation is formulated as a reachability problem on the actual service protocol LTS. Given an expected sequence of operations $\text{seq}_{\text{exp}}$ our reachability problem consists in finding a sequence $\text{seq}_{\text{act}}$ of operations on the actual service such as it is substitutable to $\text{seq}_{\text{exp}}$. The mechanism has been implemented using Zot [17], an agile and easily extensible bounded model- and satisfiability-checker. In general, Zot returns an execution trace of the specified system, which satisfies the given model. The execution trace contains a finite number of steps, each one consisting of a possible configuration of the system.

The execution trace returned by Zot is a mapping script that is then passed as input to the adapter (see Section 8 for details on the framework architecture). Each trace step contains the state in which each one of the analyzed LTS (the ones of the expected and actual services) is, the operations in $\text{seq}_{\text{exp}}$ and in $\text{seq}_{\text{act}}$ that should be invoked in that step, and the exchanged data, if any.

Considering again the services the case study in Section 2 we can assume that a designer associates to Commuter Buddy the required goal Mobility Information and tunes the code of Commuter Buddy to interact with Mobility Information Rome. Consequently the latter service will be assumed as expected service. An operation sequence will be pointed out on Mobility Information Rome, and at runtime, when the actual service will be available, if a substitutable operations sequence exists on the actual service, a suitable mapping script will be built and used to enable the substitution. Consider the operations getRoute, getRouteStops, getStopInfo as $\text{seq}_{\text{exp}}$ departing from the start state on Mobility Information Rome and ending in state infoRetrieved. A possible mapping script returned by Zot is reported in Table 7.

### 8 Framework architecture

In order to support the design process and the runtime self-adaptation of service oriented applications we built a framework based on SCENE [4], a tool that provides the support for binding external services at runtime. The runtime binding is provided through the introduction of proxies between services and their invokers. At deployment time the designed application is analyzed by SCENE and for each dock a proxy is generated and is used at runtime to mask to the application which service is actually invoked.

We tailored the architecture of SCENE to support our design process as shown in Figure 6. The framework architecture is of three layers:

- A Middleware / Service Discovery on the lower layer, providing a service registry and the facilities to publish and discover new services.
- An Adaptation Computation layer, which provides the logic needed to compute configurations and reconfigurations of service compositions, and to synthesize the needed mapping scripts.
- An execution environment for the self-adaptive service composition, which provides the facilities to run the composition code providing the possibility of performing dynamic binding and to adapt the service requests when needed.

### 8.1 The Middleware / Service Discovery Layer

The lower layer of our architecture provides service discovery facilities using a service registry based on DIRE [18], [19], a middleware offering an interface to store published services, associate them to plugs and support user queries. In this way it is possible to retrieve available service on the basis of the plug they are associated to, and consequently of the requirements they implement. DIRE also provides a publish/subscribe infrastructure based on Reds [20]. The latter can be used by clients to subscribe to the plugs they are interested to and to be notified when new services are associated to that plug. In this way the clients are informed about the available services in the registry and, through the subscription manager, they
can keep updated their service base. The middleware / service discovery layer supports the specification level introduced in the conceptual model depicted in Figure 1b.

8.2 The Adaptation Computation Layer
The adaptation computation layer provides the logic to reason on the basis of the models presented in Section 4 and 7. Consequently this layer implements, together with the execution layer explained in the next section, the composition and communication levels presented in Figure 1b.

This level features: i) An assembly calculator, which is the component in charge of solving the service building problem, to determine a binding or a rebinding when needed. The binding calculation is performed solving the service building problem, introduced in definition 1 of Section 4, through an integer linear programming solver. In our architecture we chose to use GLPK [21], an open source and freely available Integer Linear Programming problem solver. This component receives requests for binding or re-binding from the proxies, through the subscription manager, retrieves information on available services through DIRE and takes care of finding a suitable binding, solving the service-building problem. The solution computed by the assembly calculator can be used both to determine the first binding of the application or to adapt the composition in case some of the bound services are not available or the application context demands their substitution. ii) A subscription manager, which acts as an interface towards the publish / subscribe features of the lower layers. This component allows to select services in the registry through DIRE. When new services implementing a plug has to be published, the subscription manager is notified and can retrieve the service interface and protocol description, and pass them to a mapping script generator. In this way when a new service implementation associated to a selected plug is published the needed mapping script can be generated and cached by the scripts generator for later use, avoiding to make adaptation script generation a critical activity for the runtime performances of the application. Finally the subscription manager is in charge of communicating the service base composition to the assembly calculator and to retrieve the binding generated by the latter, when requested by the service proxy. iii) The mapping script generator, which takes care of synthesizing mapping scripts as explained in Section 7. The generation process starts when a new service is published in the service registry and is notified to the subscription manager. In this case the mapping script generator receives the published service interface data from the subscription manager and uses the expected service interface data that are provided by the mapping script executor. The mapping script generator is capable of caching a finite amount of generated mapping scripts and to make them available to the executor when needed. The selection of which mapping scripts should be stored is made on the basis of the quality dimensions associated to a service implementation.

8.3 The Execution Layer
The execution layer features a workflow executor, which is responsible for actually executing the workflow, a service proxy,
which is in charge of performing service invocations masking
the variability of services to the workflow executor, and a
mapping scripts executor, responsible for translating service
invocations interpreting mapping scripts, when needed. This
layer resides on client machines, together with the subscription
manager and the mapping scripts generator.

The workflow executor and the proxy present in our ar-
chitecture, are derived from the original SCENE execution
environment. However, unlike the latter, in our system various
services involved in the composition may require to invoke
other external services. Consequently a proxy should be pro-
vided for each of them. The standard SCENE environment
provides proxies only for the part of the composition executed
by a centralized orchestrator, while invoked services are treated
as black boxes. We extend this basic setting by enabling the
creation of proxies local for every dock in the composition.

Finally the execution environment includes a mapping
script executor, responsible for parsing and executing map-
ing scripts. When mapping is needed the executor requests
a mapping script to the generator in the lower layer, and
interprets the script intermediating the service invocations. An
example of mapping scripts executor at work on the case study
introduced in Figure 2, is reported in Figure 7. The latter figure
shows how the mapping script executor interprets the first three
steps of the mapping script in Table 7. Initially the executor
expects the workflow executor to perform the invocation of
getRoute on the Mobility Information Rome service. The
workflow executor forwards the invocation to the proxy which
calls the mapping script executor in order to translate it into a
valid call for getStops on Mobility Information Milan. The
response of the actual service is received by the mapping
script executor which caches part of the data returned (i.e.
stops) and forwards route to the proxy, which in turn returns
it to the workflow executor. The piece of data cached by the
proxy is used when the second operation (getStops) is invoked
by the workflow executor. In this case the getStops operation
invoked by the workflow executor requires stops as response
and the mapping script executor can return the value provided
by the previous invocation of getStops on Mobility Information
Milan, instead of re-invoking another operation on the actual
service.

9 Evaluation and Experimental Results
In order to evaluate the practical feasibility of our approach
we implemented a prototype tool to support the design and
execution of self-adaptive service compositions. Our prototype
is based on the architecture detailed in 8 and has been used to
collect experimental data and to assess the practical efficiency
through the following sets of experiments:

• A first set of experiments measuring the performances
  and assessing the correctness of the solutions produced
  by of our runtime framework on the case study presented
  in Section 2.
• A second set of experiments evaluating the time elapsed
  by the assembly calculator to solve the service building
  problem using a tile base made of a growing number of
generated service units.
• A third set of experiments evaluating the mapping scripts
generator performances.
• A fourth set of experiments evaluating the performances
  of our middleware / service discovery infrastructure.
• A fifth set of experiments comparing our runtime frame-
  work with a well known solution in literature, proposed
  by Pistore et al. in [2].

9.1 Performance Evaluation on the Case Study
In order to validate the results and the performances of
our design approach and runtime framework on a realistic
case study we tried to design an application to satisfy the
requirements of CommuterBuddy, discussed in Section 2. This
set of experiments was articulated in three parts: i) we first
tried to design the docks for the application CommuterBuddy
described in Table 1, ii) then we tried to solve the service
building problem considering as a tile base the services
reported in Table 2, iii) finally we used the mapping scripts
generator to build the needed mapping scripts for the services
bound to the composition. The docks design for the Commuter-
Buddy application was performed using the Alloy analyzer. For
each dock a model in the registry was created and we tried
to verify an assertion on it fixing a bound of 20 signatures.
Each of the verifications took about one minute of time5. We
tried to solve the service building problem using the designed
service units as $I$ and the set of services reported in Table 2
as service base $U$.

The service building problem solution reported in Table 8
is computed in negligible time and requires a minimal amount
of memory (0.6 Mbytes). We tried also to compute some
reconfigurations for that solution, keeping a part of the solution
fixed (i.e. augmenting $I$ with those service units we wanted to
be part of the solution) and varying a part of previously bound
services, and we obtained similar results. The resulting service
assemblies were analyzed by an human and found valid and
optimal with respect to the application context and available
services.

Finally we designed some protocols for the services in
Table 2 and tried to generate the mapping scripts when needed.
The protocols were inspired by services available on the
Internet (e.g. Google Maps, PayPal, Amazon Checkout or
Weather Underground). In order to create the need for some
mapping scripts we built some variations and assigned them to
some plugs and docks belonging to some services included in
the solution in Table 8. In this case the solution computing took
approximatively 0.7 seconds in the worst case and 4 Mbytes
of memory. Also in this case the produced solutions were
inspected by an human and found valid. Moreover we tried
to execute the mapping scripts using the executor introduced
in Section 8 and the invocations succeeded and returned valid
results.

This set of experiments aimed at evaluating our three-
layered approach as a whole on a realistic case study. The
results collected during this first set of experiments are quite
encouraging and suggest that on case studies taken from real

5. For a full performance assessment of the Alloy analyzer we refer the
reader to [22].
world our design approach is reasonable and our runtime framework shows acceptable performances.

9.2 Service tiles framework evaluation

In order to evaluate the performances of the composition level of our model introduced in Figure 1b, we evaluated the assembly calculator, which is its the implementation of that level in our approach. The evaluation consisted of a set of two experiments. Both the experiments had the purpose of assessing the time elapsed in the solution of the service building problem using a generated service base. In the first experiment we kept the cost vector size fixed and made the service base size and service units complexity grow (i.e. we increased the number of service unit and their complexity). In the second experiment we kept the service base size and service unit complexity fixed, and we made the cost vector size grow, to understand how the elapsed time and occupied memory vary when the cost model becomes more complex.

In order to collect data we set up the service binder on a 2.5 Ghz Intel Core2 duo machine, equipped with 4 GBytes of memory, running Linux. We tried to solve the service building problem using growing sets of service units, ranging from 60 to about 25000. The set was filled using service units with a variable number of plugs and docks, scaling up to 160 elements. Both the service units number and the plugs and docks numbers are clearly over-dimensional for a typical real world case (for instance Programmable Web\(^6\), one of the service repositories available on the web, offers about 3000 APIs), but can be used as an upper bound for our problem. The results of this experiment are reported in Figures 8a and 8b. In Figures 8a the elapsed times are reported. Figure 8a reports the elapsed time data collected in the experiment. It is interesting to notice that, even if the time growth is fast, the elapsed time to solve the service building problem with a realistic number of service units takes less than 5 seconds. The same reasoning applies to the memory occupied by the service building problem solver which for 3722 takes 160.8 Mbytes.

In the second experiment we chose to use the size of 3722 units for the service base since it is a size comparable to that of a service repository available on the web and we varied the size of the cost vector. Figures 8c and 8d report the results obtained in this experiment. Figure 8c shows that with a cost

---

vector size of 15 the service building problem takes 5.36 seconds, against the 4.8 taken to solve a problem with the same size of tile base, with a cost vector size of 5. We pushed the cost vector size up to 60 units, recording a solution time of 8.25 seconds. We conducted the same evaluation for the occupied memory, registering a datum of 17.1 Mbytes for a cost vector size of 15 units and a datum of 218 Mbytes for a cost vector size of 60.

### 9.3 Mapping Script Generator evaluation

In order to evaluate the communication level of our three-layered model we carried on three experiments on the mapping scripts generator: we created a couple of expected and actual service interface models with a fixed number states, increasing the complexity of the generated automaton by increasing the number of parameters required as input and provided as output by each operation (varying from 2 to 20) and the number of transitions outgoing from a single state (denoted by branching factor in the rest of the section and abbreviated in $bf$ in the figures of the section). We repeated this experiment for automata having respectively 10, 15 and 20 states. Also this experiment set was run on a 2.5 Ghz Intel Core2 duo machine, equipped with 4 GBytes of memory, running Linux. The purpose of this set of experiments was to demonstrate the scalability of the mapping script creation mechanism pushing the inputs of the mapping script generator up to sizes exceeding the usual dimensions of the problem found in common practice.

The elapsed times during the three experiments are reported in Figures 9a, 9b, and 9c. From the data reported in those figures two observations emerge: i) the growth of the elapsed time is influenced much more by the number of the parameters than by the growth in complexity of the automaton. This can be evinced from the closeness of the elapsed times for a corresponding number or parameters. ii) The elapsed times in the worst case (20 states, 20 parameters with a branching factor of 15) is around 60 seconds. This suggests that the approach can handle complex automata in a reasonable amount of time. The memory data show that when the branching factor grows the size of the problem also grows, as Figures 9d, 9e, and 9f show. This growth is due to the growth of clauses in the mapping script construction problem, introduced by the growth of branching factor.

### 9.4 Matching Infrastructure Evaluation

In order to evaluate the time taken by the subscriptions manager to be notified by the middleware / service discovery layer when a new service is published we ran a set of experiments on a distributed infrastructure, composed of three virtual machines each one equipped with a 550 Mhz Intel Xeon E 5530 processor and 1 Gbyte of memory, running Linux. One of the machines hosted the service registry, one the
matching infrastructure provided by DIRE and the third hosted the subscription manager. New services, featuring a number of plugs varying from 5 to 160, were published on the registry, following an uniform distribution. The subscription manager subscribed to the plugs offered by the services published by the registry and the DIRE infrastructure measured the matching times. Each of the services was published in the registry 500,000 times. The average of the matched times is reported in Table 9 along with a confidence interval of 95% for the computed matching times (i.e. 95% of the measures fall in the interval computed summing or subtracting the interval value to the average).

The purpose of this set of experiments was to evaluate the matching infrastructure showing that the average matching time is inferior to the time elapsed in building mapping scripts, while an assessment of the scalability of the registry and of the infrastructure can be found in [19], [24]. The measured times are several order of magnitudes lower than the usual time elapsed in the mapping scripts building (see Section 7 and [15], [8]). This suggests that having the subscription manager listening for new published services and caching mapping scripts is worth, as it avoids to wait for the mapping script generation to complete, in case the mapping script has been previously generated and cached.

9.5 Comparison with the Composition approach in [2]

The final set of experiments we conducted was meant to compare the performances of our runtime framework with another service composition framework available in literature. We chose to use the one by Pistore et al. described in [2] for this comparison, since this framework considers both the service composition and communication aspects, like we do in our work. In order to carry on the experiment we reproduced the experiments used as a benchmark to evaluate that approach, we recorded the times elapsed by our framework in the assembly calculation and mapping scripts generation processes and compared them with the results reported in [2].

The benchmarks reported in that article refer to the composition of a growing number of services in a choreography and are subdivided into four categories: i) composition of services without a conversational protocol (i.e. featuring only request reply interactions), ii) composition of services with a conversational binary unbalanced protocol (i.e. from each state an error state is non-deterministically reachable), iii) composition of services with a conversational binary balanced protocol (i.e. from each state two transitions depart, none of them reaching an error state), and iv) composition of services with a conversational n-ary balanced protocol (i.e. from each state n
transitions depart, none of them reaching an error state). The results of our runtime framework are reported in Table 10, where we show the elapsed time of our runtime framework on solving the service building problem and calculating the mapping script for a pair of services. It is worth noticing that the mapping script computing in our framework can take place in parallel, so the time elapsed in the mapping script creation is equal to the time elapsed by the longest script creation process, if there are enough processes working in parallel. In our experiment setting we worked under the hypothesis that three mapping script generators were available, consequently in some cases the same generator had to build more than one mapping script.

Examining the data in all the table rows, the service building problem is solved in negligible times, due to the small number of services in the service base. The first row of the table, reporting the case of simple services, records negligible times for composing the services with our approach. This is due to the fact that the services have no conversational protocol and the mapping script building step is not necessary. In all the reported cases our runtime framework outperforms the approach in [2]. This result is due to the efficiency of our mapping script generation, and to the separation in our composition model of the composition binding from the generation of mapping scripts. This results in a very terse model which leads to efficient computations.

10 Related work
The work presented in this article introduces a methodology and a runtime framework to build self-assembling, self-adapting service oriented applications. The composition step is performed through self-assembly after the services have been classified using a registry to support the process. Finally also runtime discovered services can be integrated into the system, using a dynamic mapping scripts building mechanism. Therefore our work can be related to three main topics in literature: software classification, service composition and software adaptation.

10.1 Software Classification
Software classification deals with associating a piece of software with a given description, in order to enable the piece of software automatic or semi-automatic retrieval. This is not a peculiar problem for service oriented architectures. Research on components classification has been carried on to enable their reuse by providing facilities for component search, that is, some technique to identify which components implement a specific functionality within the available ones. Many different techniques have been proposed to resolve this issue. In [25] and [26] a formal language based on algebraic specifications and a framework to reason on components specification are proposed. However these approaches require complex reasoning mechanism by means of theorem proving, making them inefficient and only partially suitable for automation.

Recently research on service oriented systems formulated the software classification problem as service discovery and proposed several approaches, which can be subdivided into approaches considering service signatures and approaches considering service behavior. In the first category we can classify those approaches that base classification on service operation signatures. Examples of works using this approach can be found in [27], [28], [29] and [30]. These approaches analyze and identifies similarity in service operation names and parameters and in service descriptions, using subtyping principles, like the one introduced in [31], or introducing the use of a domain ontology to classify documents.

In our approach we classify services relying on their behavior specification since, as stated in [32], they usually provide a better precision, which is necessary for our plugs design approach. This second category features approaches that try to establish services similarity reasoning on their specification, which can be expressed using several models. In [33] a query language for Web service discovery based on architectural specifications is proposed. This work derives queries from behavioral and structural UML design models of SOA systems. In [34] an approach classifying services on the base of their conversational protocol modeled as finite state machines, is presented. In our approach we do not rely on service protocols to classify services, since we can deal with protocol mismatches in a second time, and we prefer to rely on a descriptive specification of the service behavior an exchanged data. In [35] and [36] OWL-S and WSMO are respectively introduced. These two approaches propose to use ontologies to model both service behaviors and signatures are presented. Both the frameworks are very similar to the approach we propose in 6 and the language they define can be used in place of Alloy without any loss of expressiveness.

10.2 Service Composition
The state-of-the-art can be partitioned according to the techniques that are used to model and analyze service systems: some approaches are model-driven, others are based on optimization techniques, and others borrow from the artificial intelligence repertoire.

In model-driven approaches, the developer designs the configurations that the system can go through, possibly assisted by some generative or verification tools. Model-driven approaches work under the assumption that relevant information about system behavior and its possible reconstructions are available at design time, and can be formalized into a model which is used to guide the system through reactions due to component failures or changes in the application context. [37] presents the PLASTIC approach to develop self-adaptive services able to respond to changes in the application context (defined by users preferences) and to maintain certain levels of quality-of-service. With PLASTIC, the system is designed by building a model. From this model, several variants of the service code can be generated. Different variants are used to support adaptation in response to changes of context or quality-of-service requirements. The current PLASTIC framework provides adaptation at discovery time (i.e., the deployed application is customized at binding time) but not at runtime.

The service-tile model frames the compositional aspect of building self-adaptive service-oriented systems and enables reconfiguration at runtime. Consequently, an extension of the
PLASTIC framework with service tiles would allow applications to be reconfigured both at design time and at runtime. In [38] and [39] two model-driven approaches are presented. They require to include all the possible variation points (i.e., aspects in which the system can change its configuration in order to perform adaptation) in the model at design time. This limits the situations the system can react to at runtime, since the number of configurations that need to be explicitly enumerated at design time can be hardly handled by a human developer.

Our Work also proposes a model-driven approach to design self-adaptive service oriented applications, focusing on reducing the amount of information to be elicited at design time. In fact, service-tile models allow to build a system designing a part of it (i.e. the initial set), while the rest of the system is determined using the self-assembly capability of the tile-based system. As showed in the thesis, this technique can be used also to find possible reconfigurations for the application, without having the designer to explicitly enumerate them at design time.

Optimization-based techniques, such as linear programming or genetic algorithms, are used to select a proper assembly of services to be invoked. At runtime, if one or more services fail or the quality of service of the composition becomes inadequate re-optimization is exploited to react to the situation. Linear programming based approaches can be found in [40] and [41]. These approaches model the service selection problem as an optimization problem: they model a workflow in terms of the operations it requires, define a complex quality-of-service model, and solve optimization problems to find the assembly of services to be allocated for each of the tasks, so as to maximize the overall quality-of-service. Service-tile models are also implemented using integer linear programming; however, their focus more on providing a flexible tool to design self-adaptive service-oriented systems than on optimizing the binding according to a sophisticated quality-of-service model.

In [42] the authors present a genetic-algorithm-based and quality-of-service-constrained service selection mechanism. In this approach, binding of a service-oriented system with a set of services, which meet some non-functional constraints, can be performed at runtime. This solution allows computing and reconfiguring only the portion of the system affected by changes. Unfortunately, genetic algorithms are usually computationally expensive. This prevents their usage at runtime for dynamic reconfiguration.

Artificial intelligence based techniques are also used to determine a proper service assembly, typically by encoding service selection as a planning problem. Both the approaches in [43] and [2] planning techniques are used to determine which service assemblies can fulfill some goals specified by a developer at design time. The reconfiguration can take place both at design time and at runtime, allowing also subsets of the system to be reconfigured. The former approach depends on the hypothesis that a centralized executor will orchestrate the process execution and will take care of reconstructions when needed. Applications built under this hypothesis fail to provide suitable reconstructions in scenarios similar to that presented in Section 2 where invoked services are needed to reorganize autonomously. Service tiles overcome this limitation modeling system component services in terms of offered and required operations. The second approach also overcomes the centralized executor hypothesis and proposes a planning technique capable of producing service choreographies, just like our approach. Anyway in Section 9.5 of Section 9 we showed that the approach is outperformed by our runtime framework.

[44] also specifically addresses the problem of building component-based applications, with a goal-oriented approach. Components are selected by adaptive planning (i.e. by a selection strategy that varies in response to the system’s variability). This solution, however, does not focus on the distributed application domain, while the Service-Tiles model is more flexible in this respect.

### 10.3 Software Adaptation

The approaches that support service adaptation can be categorized into those that require human intervention in the definition of mapping scripts or equivalent mechanisms, and those that offer some automatic tool.

Among the approaches in the first category, we mention here the ones in [45] and in [46] which offer a development methodology for a developer to manually develop adapters, and those in [47] and [48] which offer a model checking based approach to detect service conversational protocol inconsistencies, but demand to a human developer the task to solve them. The work in [49] describes a semi-automated tool capable to assist humans in adapters development by providing hints about possible protocol adapters, but still relies on a developer to take decisions in complex cases. In our work mapping script generation is needed to support runtime substitutions of services. For this reason we need to rely on automated approaches.
Automated approaches try to solve this issue by generating adapters that are inferred from specifications associated to services or trying to restrict the service behaviors in such a way that protocol mismatches are avoided. Generative approaches solve the problem by generating adapters which act as mediators between the invoker and the service. Some of these approaches are based on the use of ontologies. Among the others, our previous work [16] and [50] exploit a domain ontology (specified in SAWSDL)\(^8\) to annotate service interfaces. At run-time, when a service becomes unavailable, a software agent generates a mapping by parsing the ontological annotations in the interfaces. SAWSDL\(^8\) document, which represents the service state. When an invoked service becomes unavailable, SCIROCO exploits the SAWSDL annotations to find a set of candidates that expose a semantically matching interface. Then, the WS-ResourceProperties document associated to each candidate service is analyzed to find out if it is possible to bring the candidate in a state that is compatible with the state of the unavailable service. If this is possible, then this service is selected for replacement of the one that is unavailable. All of these three approaches offer full run-time automation for service substitution, but can address only those mismatches that concern data and operation names while they disregard those concerning the interaction protocol. In [52] a generative approach based on a subset of \(\pi\)-calculus is introduced. The approach is capable of dealing with both interfaces and protocols adaptation, but it requires a developer to provide a mapping between messages and data exchanged by services, in case their names differ. Our approach requires a similar mapping in case of mismatches between service operation names and parameters, but we are able to automatically derive it from the specification level of our development model (see Section 4 and 7). Moreover our approach can be considered both generative and restrictive, since it generates mapping script which restrict the behavior of the actual service and are interpreted by an adapter working as a mediator.

Restrictive approaches solve mismatches by avoiding behaviors that can lead to deadlocks. In [53] a restrictive approach which proposes a formal framework and a tool to build adapters is presented. This approach starts from the capability of a service integrator to provide adaptation contracts (i.e. mappings between names of different operations) and translates the conversational protocols of services involved in the adapter construction into a petri net model. Using this model the approach can handle complex cases of protocol mismatches. The approach in [54] aims at creating adapters by enforcing a some desired properties out of a set of behaviors exhibited by the services. This approach starts from the behavior of services to adapt and from the property to enforce, specified as a Message Sequence Chart [7] and, through model checking techniques, generates some code cutting off those behaviors not verifying the desired property. This work was also extended in [55] to enable adapters to enforce some QoS constraints. The aforementioned works are different from ours because they consider that the service protocol is composed by a set of messages, but they neglect the data-flow constraints. Our approach considers them to build mapping scripts, because services interactions are mainly based on the exchange of data [56]. Moreover both [53] and [54] develop global adapters considering a component service at whole, disregarding which part the client is actually using. This aspect is limiting when the possibility for the system to evolve by replacing components at runtime has to be taken into account. In our work the mapping script construction is needed for enabling runtime replacement of components, therefore we build adapters only for those parts of the service a client is using during the interaction. This makes the mapping scripts building process faster and more suitable to be executed at runtime.

Approaches that similarly to ours develop adapters only for the part of the service a client is using are presented in [57] and in [58]. These approaches offer similar features to ours and the second of them has been implemented in an open source tool.\(^{10}\) While both these approaches appear to fulfill our need for supporting interaction protocol mapping, they may present some shortcoming in terms of performances due to the high cost of exhaustive graph exploration algorithms that could prevent their usage in on-the-fly mapping derivation. While no data about performances are available for the approach in [57], we could exploit the tool offered by [58] and in [8] we showed that the mapping scripts building time required by their tool is remarkably higher than ours in most of the cases.

11 Conclusions

Service oriented technology proposes a new applications design paradigm which allows for building complex applications from simpler components possibly issued by third parties. Self-adaptive service oriented systems allow to reconfigure service oriented applications to respond to failures or unavailabilities of component services or to allow the application to adapt to changes in the context. Research proposed many frameworks that try to add self-adaptiveness to service oriented applications, addressing mainly the technological level. Unfortunately there are still some barriers that prevent the wide adoption of this technology. In fact developing self-adaptive service oriented systems requires the developer to cope with a greater level of uncertainty, which limits his knowledge about the application. This makes the traditional development process unsuitable, as it clearly separates a design time phase in which the developer specifies all the information and a runtime phase in which the elicited information is used.

In this article we solved the aforementioned issues, developing a design methodology and a framework to overcome. The main contributions of the work are the following: i) A three levels application development model which clearly separates the specification level, in which the developer specifies the requirements his artifacts implement or demand to external

services, a composition level, in which the information elicited at the specification level is exploited to assemble services and an interface level, which takes care of enabling communication between component services assembled at the composition level. ii) An efficient composition framework based on a self-assembly technique called Service-Tiles, which supports the composition level of our three levels model. iii) A registry based design technique for service units, used in Service-Tiles. This technique supports the specification level of our model. iv) A runtime model-based mapping scripts synthesis technique, which supports the communication level of our model. The outcome of the work has been implemented in a prototype framework and evaluated on some input sets, taken from real world applications, generated ad-hoc to stress the techniques scalability or taken from similar approaches in literature, in order to build a performances comparison. The experiments results showed that our framework shows performances compatible with its runtime usage and outperforms other similar frameworks previously proposed in literature.

The research work proposes the basis for two future research work directions: a) In this work we developed a formal framework and a prototype to enable the dynamic integration of unforeseen services. The integration is not always possible, since an adapter can not be developed for all pairs of interfaces. The first open research idea coming out of this article is to develop a technique that, given the specification of a service, can be used to learn which interfaces the service is capable to interact with. b) This work considers services showing input/output dependencies between their operations (i.e. showing a conversational protocol), but is currently limited by the hypothesis that services do not have an internal state. Considering also stateful services is a challenging task, since it requires a proper representation of status information and a formalisms to efficiently reason about the substitution process.

ACKNOWLEDGMENTS

This research has been partially funded by the European Community’s FP7/2007-2013 Programme, grant agreement 215483 (S-Cube), IDEAS-ERC Programme, Project 227977 (SMSCom), and MIUR PRIN project "D-ASP: Dependable Adaptable Software Architecture for Pervasive Computing".

REFERENCES


A Frame of Reference for SOA Migration

Maryam Razavian* and Patricia Lago

Department of Computer Science, VU University Amsterdam, The Netherlands
{m.razavian,p.lago}@few.vu.nl

Abstract. Migration of legacy systems to service-based systems constitutes a key challenge of service-oriented system engineering, namely rehabilitation of pre-existing enterprise assets while conforming to service engineering principles. Over a decade there has been an increasing interest in the approaches addressing SOA migration. These approaches mainly differ in ‘what is migrated’ and ‘how the migration is performed’. Such differences aggravate achieving a general understanding of ‘what SOA migration entails’. To solve this problem, we conducted a systematic review that extracts main migration categories, called SOA migration families, from the approaches proposed in the research community. Based on the results of the systematic review, we describe eight distinct families along with their characteristics and goals. These families represent a first frame of reference for SOA migration which brings order and enhances understanding on how migration can be carried out.

1 Introduction

One of the key promises of service oriented paradigm is facilitating reuse of enterprise assets in legacy systems [1]. Migration of legacy systems to service-based systems enables achieving advantages offered by SOA while still reusing the embedded capabilities in the legacy systems. Since the early use of SOA, migration of legacy systems to SOA has caught a lot of attention. Various studies present an approach for such migration. These studies mainly differ in the way they provide solutions for two challenging problems of what can be migrated (i.e. the legacy elements) and how the migration is performed (i.e. the migration process). As an example, some studies address implementation aspects of migration by providing methods for altering segments of the legacy code to web services. On the other hand, other studies focus on refactoring the legacy architecture to a service-based architecture based on business drivers such as business rules, benefits and risks. Such differences can hinder achieving a general understanding of ‘what SOA migration entails’ and therefore making it difficult to determine how to migrate.

* This research has been partially sponsored by the Dutch Joint Academic and Commercial Quality Research and Development (Jacquard) program on Software Engineering Research via contract 638.001.206 SAPIENSA: Service-enAbling PreexIsting ENterprISe Assets; and the European Community’s Seventh Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).
To obtain such understanding, we conducted a systematic review that extracts main migration categories existing in the field. Due to its methodological rigor, we chose systematic review as our research method in aggregating existing SOA migration approaches. Furthermore, the strength of systematic reviews in minimizing the bias in the review process enhances the extraction of sound and meaningful migration categories. By devising a coding procedure, we analyzed the studies and extracted eight distinct categories. As an answer to the question of ‘what SOA migration entails’, this paper describes these eight categories, called SOA migration families. Using a holistic conceptual framework that reflects distinct conceptual elements involved in the migration process, SOA migration families are typified in a unified manner. As such, these families act as a frame of reference for SOA migration which brings order and enhances understanding in how such migration can be carried out. Accordingly, this frame of reference increases awareness of the ways in which a legacy system can be migrated to SOA.

2 Research Method

We followed a formal systematic literature review process based on the guidelines proposed in [2,3]. As part of the process, we developed a protocol (described in the following) that provided a plan for the review in terms of the method to be followed, including the research questions and the data to be extracted.

2.1 Review Protocol

Research Questions. In order to answer the question of ‘what SOA migration entails’, we seek for extracting the types of SOA migration approaches regarding their solution for migration. The systematic review envisions providing an evidence base of existing SOA migration approaches and further categorizing them. To achieve this goal, we define the following research questions:

What methods/techniques/processes/approaches regarding legacy to SOA migration, have been proposed in research community so far? In particular, the following aspects facilitate characterizing the approach: (a) what are the activities carried out? (b) what artifacts are used or produced? (c) what are the knowledge elements used within different activities?

Search Process. As the first step of systematic search, three main keywords are built from our research question, namely: migration, legacy systems and SOA. Considering the related terms for the keywords, we defined the following search string:

(SOA or ‘service-oriented’ or ‘service-computing’ or ‘service-based’ or ‘service-centric’ or ‘service’ or ‘service-engineering’ or SOSE) AND (‘legacy code’ or ‘legacy system’ or ‘existing system’ or ‘legacy component’ or ‘existing code’ or ‘existing asset’ or ‘existing component’ or ‘pre-existing code’ or ‘pre-existing system’ or ‘pre-existing component’ or ‘legacy software’ or ‘existing software’ or ‘pre-existing software’) AND (migration or modernization or transformation or
reengineering or re-engineer or evolving or reuse or ‘service mining’ or ‘service identification’ or ‘service extraction’)

We used the following libraries as main resources: IEEE Explore, ACM Digital Library, ISI Web of Knowledge, SpringerLink, ScienceDirect, and Wiley Inter Science Journal Finder. As major venues on service-oriented systems like ICSOC started in 2003, we decided to set 2000 as the start date to minimize the chance of overlooking relevant studies. We applied the search terms to titles and abstracts considering that they provide a concise summary of the work. This decision was assessed by running the search string on data sources and checking if the pilot studies are retrieved.

Selection of Primary Studies. Peer-reviewed articles in the field of software engineering that satisfy the following inclusion criteria are selected as a primary study. I1) A study that is about migration to services. Rationale: studies which support migration to other types of target systems (not to service-based) should be excluded. I2) A study that addresses migration from existing legacy assets. I3) A study which proposes a solution for migration. Rationale: studies that not specifically provide a solution for the migration problem should be excluded. For instance, studies presenting challenges on SOA migration are out of scope of this work.

2.2 Data Analysis

As mentioned, this study seeks for achieving a general understanding of ‘what SOA migration entails’ by categorization and comparisons of the approaches. The question that we faced is how to systematically analyze the primary studies, in such way that the meaningful categorizations are determined. We chose coding as our qualitative analysis method, since we were seeking for the conceptualization of data, not actual data per se. According to [4], one method of creating codes is to have an initial set of codes, called ‘start-list’, that is refined during the analysis. Our start-list stems from a SOA migration framework (called SOA-MF), proposed in earlier work [5]. The comprising conceptual elements of SOA-MF, described in Section 2.3 provide pieces of information to position each migration approach into SOA-MF. Hence, by coding the primary studies their associated mappings on SOA-MF is achieved. Similar to searching process, in order to carry out the analysis systematically, its procedure has to be made explicit. Inspired by the procedure proposed by Lincoln and Guba [4], we devised the following coding procedure for our purpose: (1) Filling in/Surfacing activities involved in migration: coding activities and refining the codes labeling activities, identifying the new activities. (2) Filling in/Surfacing process: coding the inputs and outputs of an activity, identifying the sequence of activities. (3) Surfing knowledge elements: Identifying the knowledge elements as well as the level of abstraction in which they reside. (4) Bridging/Observations: Identifying patterns, goals and gaps in the migration process.

1 We cannot be certain that we have covered all studies with a publication date in 2009, since studies may not have been indexed yet at the time we conducted our review (Jan 2010).
2.3 The SOA Migration Framework

The SOA migration framework [5], called SOA-MF, is a skeleton of the holistic migration process along with the distinct conceptual elements involved in such a process (see Fig. 1.I). The framework consists of three sub-processes: reverse engineering, transformation and forward engineering. SOA-MF follows a horseshoe model by first recovering the lost abstractions and eliciting the legacy fragments that are suitable for migration to SOA (reverse engineering), altering and reshaping the legacy abstractions to service based abstractions (transformations), and finally, renovating the target system based on transformed abstractions as well as new requirements (forward engineering). Reverse engineering starts from existing implementation and continue with extracting the design entities (code analysis), recovering the architecture (architectural recovery) and recapturing abstractions in requirements or business models (business model recovery). Within the transformation sub-process the activities of design element transformation, composition transformation and business model transformation, respectively, realize the tasks of reshaping design elements, restructuring the architecture and altering business models and business strategies. The forward engineering sub-process involves the activities of service analysis, service design and service implementation. Finally, the framework covers different levels of abstraction including concept, composite design element, basic design element and code.

3 Results

By applying the search query defined in Section 2.1 to the selected data sources, we obtained 258 articles whose titles or abstracts contained the keywords specified in the search query. After applying the inclusion/exclusion criteria, 51 were considered relevant for our study. Although we identified 51 articles by this search process, some articles were earlier or short versions of other articles. Thus, we ended up with 44 unique primary studies. Using the procedure explained in Section 2.2, we coded the primary studies and consequently obtained their mappings on SOA-MF. Although initially we had 44 primary studies, the analysis resulted in 39 different mappings on SOA-MF. This was because in five studies the description of the migration approach was too general or vague, for us to be able to codify them. By thoroughly analyzing the mappings, we identified a set of meaningful relationships among the approaches with similar coverage patterns and their migration objectives and solutions. More precisely, thanks to SOA-MF, the migration approaches that pursue a common migration goal using conceptually similar activities and artifacts, have graphically similar coverage pattern as well. Accordingly, by considering similar SOA-MF coverage patterns, out of 39 different mappings eight distinct families of SOA migration approaches were extracted. Fig. 1.III illustrates the schematic form of distinguished mappings that are dedicated to each family. As an example, F4.b is a schematic form of the mapping shown in Fig 1.II. Section 3.1 describes each family in the following way: 1) the family at a glance provides a general description of the
implications of each family, illustrated in Fig. 1.III 2) the observations include analytical explanation of ‘what migration entails’ in each family.2

3.1 Families of SOA Migration Approaches

Code Translation Family (F1). Example Members: [6,7]. Family at a glance: The mapping of the approaches of this family on SOA-MF (simplified in Fig 1.F1(a)), reflects the following feature: out of the three sub-processes, the migration process is limited to transformation at system level in which the existing legacy code is transformed to service-based implementation.

Observations: Migration, in code translation family, entails moving the legacy system as a whole to a service-oriented platform or technology, without decomposing the existing system. We identified two main categories in this family: (1) translating the whole code to web services, and (2) wrapping the whole application as a web service. The problem addressed by the first category is to translate a legacy code to a web service implementation. The second category embraces

---

2 Due to space constraints, not all members of each family are introduced. The full list of the primary studies distributed among the families is available at http://www.few.vu.nl/~mrazavi/SOAMigrationAppendix.pdf
encapsulating the interfaces of the existing application to a (web) service interface. In our view, this category is in line with classical black-box reengineering techniques, that integrate systems via adaptors and allow the application to be invoked as a service. We also found that the members of this category are common in altering the legacy interaction protocol from synchronous communication to asynchronous communication, mostly using Enterprise Service Bus (ESB).

Service Identification Family (F2). Example Members: \[8,9,10,11\]. Family at a glance: In this family, the transformation sub-process is not covered, meaning that reshaping of the legacy elements to service-based elements is not realized. The reverse engineering sub-process is carried out in all family members, while forward engineering occurs only in some (Fig 1.F2(c,d)). Reverse engineering, here, embraces the activities of ‘code analysis’ and ‘architectural recovery’ and forward engineering is limited to the ‘service analysis’ activity.

Observations: This family of approaches focuses on the identification of the candidate services in the existing legacy system. Migration, here, is limited to ‘what is migrated’, whereas ‘how migration is performed’ is not addressed. The legacy elements realizing the candidate services are identified using reverse engineering techniques. This implies that, by extracting system’s functionality, structure, or behavior, the legacy elements suitable for migration are identified. Based on the criteria guiding the service identification, we extracted two main categories in this family: in the first category, (Fig 1.F2(a,b)), which just covers the reverse engineering sub-process, the identification of the candidate services is guided by a set of constraints characterizing a service (e.g., \[8,12\]). These constraints are inherently features, requirements or properties characterizing a service or service composition. As an example in \[8\], higher level representations of the legacy system are extracted aiming at identification of legacy elements (e.g. components, modules) that are ‘cohesive’ and are ‘loosely coupled’. The second category (Fig 1.F2(c,d)) mostly focuses on locating a specific business functionality in the legacy code. While sets of constraints drive the service identification in the first category, it is a set of desired business functionalities that guide this task in the second category (e.g., \[9,10,11\]). The business functionalities are codified using sets of structural representations such as business ontology or business service model. We found that service identification, here, is realized by mapping the desired business functionalities (extracted within forward engineering sub-process) on a model representing the legacy system.

Business Model Transformation Family (F3). Example Members: \[13,14,15\]. Family at a glance: In this family, the reverse engineering and the forward engineering sub-processes are not covered. This means that capturing the abstractions of the existing legacy system as well as development of the service based system is not addressed by these family members. Migration is realized by the transformation sub-process, carried out at concept level.

Observations: Based on the types of transformation at concept level, we found two main categories in this family: (1) approaches providing a meta-process for
migration, e.g., [14,15], and (2) approaches with business process reengineering e.g., [13]. The main goal of the meta-process category is to support the decision regarding ‘how to perform migration’. The constituent activities of these approaches support decision making on the migration process itself. Due to its orthogonal view on the migration process, we recognize this category as a ‘meta-process’. a) information representing the As-Is state of the legacy system (i.e. existing capabilities and constraints) and b) information about the To-Be state of the service based system (i.e. target capabilities and constraints). The output, extracted based on a comparison among the As-Is and To-Be states, constitutes a roadmap shaping the migration process.

Despite having the same coverage pattern on SOA-MF, we found that the business process reengineering category of this family reflects a different perspective on SOA migration: altering the business process of the existing legacy system to serve as a basis for top-down service development. We found that the main focus of this category is quite similar to business process management techniques that address reengineering the existing business processes based on the new requirements and goals. The difference among these two categories motivates a refinement to SOA-MF that is further discussed in Section 4.

Design Element Transformation Family (F4). Example Members: [16,17,18]. Family at a glance: According to coverage of this family members on SOA-MF, the common feature is that the transformation sub-process only occurs at ‘basic design element’ level. Similarly, reverse and forward engineering sub-processes, if covered, are also limited to this level.

Observations: Migration in this family is limited to reshaping the existing legacy elements to the service-based elements. More precisely, a set of legacy elements, extracted by means of the ‘code analysis’ activity or simply known beforehand, are transferred to a set of services or service-based elements. For instance, a ‘component specification’ is altered to ‘service specification’ [17], or a ‘module’ is reshaped to a ‘service’ [18] or ‘segment of code in the persistence layer’ is transformed to a ‘data service’ [16]. We found that the understanding of the existing system proceeds until its constituting structural (e.g. components or modules) or behavioral elements (e.g. interaction models) are extracted. Afterwards, these elements are reshaped to the associated element in the service-based system, and further implemented. Most of the wrapping approaches fall in this family.

Forward Engineering with Design Element Transformation Family (F5). Example Members: [19,20]. Family at a glance: This family fully covers the forward engineering sub-process, whereas transformation and reverse engineering sub-processes occur at ‘basic design element’ level.

Observations: The main focus of this family is on development of service-based systems starting from the desired business processes. These processes need to be designed in terms of constellations of interacting services obtained from both ‘migrated’ and ‘newly designed’ services. Here understanding of the existing
legacy system (within the reverse engineering sub-process) is limited to locating the required functionalities in the target service based system. The migration here entails Top-Down service-based development while locating the realization of the required business functionalities and transforming them to services.

Design and Composite Element Transformation Family (F6). Example Members: 21, 22, 23. Family at a glance: The three migration sub-processes occur in the two levels of the ‘basic design element’ and ‘composite design element’, meaning that the members include both design element and composition transformations.

Observations: What characterizes this family is having transformation in both levels of ‘basic design element’ and ‘composite design element’. This entails altering legacy elements to services (i.e. design element transformation) as well as reshaping the structure and the topology of legacy elements to realize new service compositions (i.e. composition transformation). Prior to refactoring the architecture and reshaping the legacy elements, the legacy architecture is recovered during reverse engineering sub-process. Architectural recovery, here, is guided by a set of service-specific constraints (e.g. high level of granularity, autonomy) and the desired functionalities (e.g. business services). We also observed that refactoring architecture (i.e. composition transformation), triggers the lower level transformations (i.e. design element transformation and translation). As shown in Fig. 1.F6, the forward engineering sub-process is not similarly covered in all members. In some members 23, 22 (Fig. 1.F6(b)), forward engineering includes both design and development of the services based on the service compositions, extracted from architectural refactoring. While, in other members such as 21 (Fig. 1.F6(a,c,d), this sub-process is limited to implementation of the wrapped services. To sum up, migration embraces recovering and refactoring of the legacy architecture to the service-oriented architecture as well as reshaping the legacy elements to service-based elements.

Pattern-Based Composition Transformation Family (F7)
Example Members: 24, 25. Family at a glance: Migration only includes the transformation sub-process at ‘composite design element’ level. This implies that the architecture of the existing system is altered or configured into the service based architecture.

Observations: A common feature in this family is using ‘patterns’ for transforming the existing architecture to service-based architecture. Patterns are inherently reusable solutions, that are here used to extract services or facilitate transformations of legacy elements to services. We found two main types of patterns used by the members of this family: application-specific (e.g. reference architectures or workflow patterns) and application-generic (e.g. Facade, Mediator, and Chain of responsibility) used. The application-specific patterns mainly represent common structure of a domain, while workflow patterns represent business or workflow processes in an application domain. We argue that the recurring nature of the transformation problem, initiated the need for patterns.
that codify reusable solutions. *Migration here entails pattern-based architectural transformation to SOA.*

**Forward Engineering with Gap Analysis Family (F8). Example Members:** [20]. **Family at a glance:** The transformation sub-process, in this family, occurs in the three tiers of ‘concept’, ‘composite design element’ and ‘basic design element’. As shown by (Fig. 1.F8(a)), the forward engineering subprocess covers the activities of ‘service analysis’ and ‘service design’ whereas the reverse engineering sub-process is not covered.

**Observations:** This family mainly focuses on top-down service development, starting from extraction of the business model of the target system and further designing service compositions and services. What distinguishes this family from pure top-down service development approaches is that at each abstraction level (including concept, composition and design level) a comparison (a gap analysis) among the new and pre-existing artifacts occurs. This comparison serves to assess how the desired business services can be realized by exploiting pre-existing capabilities. We observed that the gap analysis at the highest level indicates the discrepancies among business model the existing and the target system. This comparison further triggers design decisions regarding how the new service functionality can be obtained by reusing or revamping pre-existing assets. This implies that transformation at the highest level guides the transformations at composition and basic design element levels. *The migration here entails Top-Down service development while assessing the reuse opportunities in all abstraction levels.*

4 Discussion

In the following we discuss the possible threats to validity of our analysis and our findings.

**Threats to Validity.** One of the threats to validity of the study is that the review is mainly conducted by a single researcher. However, subjective interpretations are mitigated by both following a systematic protocol, checked and validated by senior researchers experienced in software engineering, systematic reviews and SOA and validating it further using a pilot study. Additionally, we explicitly included only publications whose objective is to present a solution for migration. It is possible that a publication proposes also a solution for migration blended with other objectives, so that the contribution on migration is not clearly represented. To mitigate this threat, we added some more generalized keywords such as ‘reuse’ in the search terms.

Threat to validity of the analysis is in the general applicability of the codes used for characterizing and classifying migration approaches. An assuring factor in this regard is that the start-list of codes is extracted from a conceptual framework published in a service-oriented computing forum, after being peer reviewed by experts in the field [5]. This framework stems from existing theory on
reengineering and architectural recovery while it is constantly refined through our coding procedure. This further consolidates its general applicability.

Finally, the scope of our review is restricted to the scientific domain. The threat here is that very relevant migration approaches originated in industry, if not described in scientific publications, are not covered. To fill this gap, we are carrying out a survey focusing on how SOA migration is performed in industrial practice.

**Refinements to SOA-MF.** As described in Section 2.2, code refinement constitutes a key part of our coding procedure. As codes stem from the original SOA-MF, refinements to code entails changes to SOA-MF as well. In other words, the need for refining the codes reflects the relevant changes to the framework which enhance it’s ability for characterizing different approaches. Below, two examples of the findings motivating refinements to SOA-MF are presented: 1) In the initial version of SOA-MF framework [5] the ‘code translation’ activity did not exist, since it was originated from the existing theory on architectural recovery [27] that did not consider transformation at code level. During the early stages of this analysis, we observed that there are some migration approaches that translate the legacy system code to web services. In order to characterize such approaches a code called ‘code translation activity’ was added. Eventually, the ‘code translation activity’ was added to the transformation sub-process of SOA-MF as well. 2) As mentioned, we observed that the two categories of ‘meta-process’ and ‘business process reengineering’ in the business model transformation family, are so different in their objectives that do not resemble members of the same family. This suggests that the current framework does not fully highlight their differences. Although the ‘meta-process’ category is positioned on the concept level, its ‘meta’ characteristic suggests the need for an orthogonal dimension on the current two-dimensional SOA-MF. The third dimension reflects the decision making process, orthogonal to migration process itself. This motivates further research on refining SOA-MF to a three-dimensional framework.

**Synthesis of Findings.** In summary, our study yields two main findings:

(I) The description of the families suggests the following two themes in the objectives of SOA migration approaches: modernizing the legacy system and facilitating reuse during service-based development. 1) Migration for modernization: In a set of families (i.e. F1, F2, F4, F6 and F7) the migration aims at renovating the existing legacy system to reconstitute it in the new form of SOA. Consequently, they mainly focus on how to adapt the legacy systems to the SOA environment. To this end, the reverse engineering sub-process realizes understanding of the existing system, transformation sub-process specifies how to restructure the legacy systems, while forward engineering sub-process realizes the restructuring. 2) Migration for reuse in service-based development: in some families (i.e. F5 and F8) the main goal of migration is to facilitate reuse in building new service-based systems. This goal changes the order in which the three sub-processes are carried out. Accordingly, the forward engineering sub-process realizes the service-based development; to do so, the reverse engineering
sub-process facilitates identifying reusable legacy assets and the transformation sub-process reshapes the legacy elements to service-based elements. Note that F3 provides a ‘meta process’ view on the whole migration approach that pertains to both themes and is therefore not categorized in the two themes. Identifying these two themes can help the designers better select the appropriate approach for SOA migration.

(II) Considering the abstraction levels, SOA-MF presents four levels of ‘code’, ‘basic design element’, ‘composite design element’ and ‘concept’. The ‘code’ level deals with the actual running system, the ‘basic design element’ and ‘composite design element’ levels represent the design solution, whereas the ‘concept’ level represents the problem. In a number of primary studies, design solution artifacts reflect two layers of atomic element (e.g. modules), and composite element (e.g. architecture). In the same vein, we expected to observe the similar layering scheme for the concept level artifacts. For instance, having two layers of business services (i.e. atomic element) and business processes (i.e. composite element). This layering scheme, for example, has been leveraged in [1], in which business services play a mediator role to articulate business processes, with the underlying solution services. However, only one of the primary studies distinguished the abstraction levels in the concept level [26]. This motivates further research since the importance of having different abstraction levels at ‘concept level’ has been considerably acknowledged in practice.

5 Conclusion

It is hard to gain insightful understanding of how to perform migration in an emerging and still fuzzy research field like SOA migration. To reach such understanding, this paper has presented a frame of reference for SOA migration by defining eight distinct families. To identify these families, we mapped each study on SOA-MF and clustered them into different families based on the similarity of their mappings. By analyzing the migration approaches, a more profound description for migration families was obtained that characterizes each family from the two following aspects: what is the main objective of migration and how this objective is pursued by different categories of solutions. By describing the families, this paper brings order on the existing SOA migration approaches and consequently provides a “bird’s-eye” view of ‘what SOA migration entails’.

By positioning a migration approach on these families, insight in the following aspects can be achieved: to what extent the reverse engineering, transformation and forward engineering occur, what activities are carried out, what artifacts are used or produced, what abstraction levels are covered, what is the main objective of migration, and finally, what are the available solutions.

During the course of the systematic review, we observed different types of knowledge that guide each of reverse engineering, transformation and forward engineering sub-processes. As our future work, we will further analyze the results of this study to identify the types of knowledge driving SOA migration. In addition, we are conducting a follow-up study targeting at SOA migration
approaches in practice to examine what migration families are used in practice and what is the gap among the migration families in academia and industry.

References

Service Identification Methods: A Systematic Literature Review

Qing Gu* and Patricia Lago

Department of Computer Science, VU University Amsterdam, The Netherlands

Abstract. Many service identification methods (SIMs) have been proposed to support the determination of services that are appropriate for use in an SOA. However, these SIMs vary in terms of analysis objectives, identification procedures and service hierarchies. Due to the heterogeneity of the SIMs, practitioners often face the difficulty of choosing a SIM that copes with available resources and fits their needs. To gain a holistic view of existing SIMs and to support the selection of the right SIM, in this paper we present the results of a systematic literature review. A total number of 237 studies were examined, of which 30 studies were selected as primary studies. From these studies, we identified different types of inputs, outputs and processes used by the existing SIMs. Based on these results, we created a matrix which can be used in three different ways for practitioners to select among alternative SIMs.

1 Introduction and Research Questions

Service-oriented architecture (SOA) is an architectural enabler for quick response to business changes and effective reuse of software assets [1]. In an on-demand world, many enterprises have adopted SOA in order to gain competitive power [1]. A key factor that determines whether an enterprise really can benefit from adopting SOA is the design of services, including the scope of functionality a service exposes to meet business needs, and the boundaries between services to achieve maximum reusability and flexibility [2].

In the design of services, service identification (SI) is a significant task aiming at determining (based on available resources) services that are appropriate for use in an SOA. Many service identification methods (SIMs) have been proposed from both academia and industry. However, these SIMs differ significantly, ranging from source code extraction to business domain analysis, from bottom-up to top-down strategy and from ontology-based process to guideline-driven process. Accordingly, the inputs and outputs of these approaches vary as well. Some approaches start with business process whereas some others start with domain knowledge (e.g. goals and strategies); some approaches focus on business services whereas others focus on software services.

* The research leading to these results has received funding from the European Community’s Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).
Given many SIMs, a common question that practitioners face is how to select the most appropriate SIM that copes with the available resources and fits their needs \[2\]. Some enterprises, for instance, not only have well defined business process models in place but also well documented goals, business partners and enterprise capabilities (e.g. IT resources) to support its business process. For them, a SIM that takes all of this information into account will be more suitable than those identifying services only based on e.g. business processes.

Despite the comparisons (e.g. \[3,4\]) presented so far, none systematically searches for all the existing SIMs. As a result, a holistic overview of extant SIMs is missing. Moreover, the criteria used in the existing comparison frameworks cover many aspects, including economic, business, and technical aspects. However, a comparison of the basic elements (such as the inputs, outputs and processes) of the SIMs is currently missing. When selecting SIMs one often starts questioning what is required for using certain SIMs, what can be expected from them and how to carry them out, before addressing other requirements. Without such an overview of the basic elements of SIMs, the selection of SIMs becomes more complicated. Accordingly, the following research questions arise:

- \textit{RQ1}: What are the existing SIMs?
- \textit{RQ2}: How do the SIMs differ? This can be elaborated into: \textit{RQ2.a} what types of inputs do the SIMs start from? \textit{RQ2.b} what types of services do the SIMs produce? \textit{RQ2.c} what types of strategies and techniques are used by the SIMs?

To answer these research questions, we decided to conduct a systematic literature review, which are particularly powerful in collecting and analyzing existing work. It can maximize the chance to retrieve complete data sets and minimize the chance of bias and summarizing the existing evidence \[5\]. In the remainder of the paper, Sec. \[2\] reports on the review results; Sec. \[3\] presents an input-output matrix that aids the selection of SIMs; and Sec. \[4\] concludes the paper.

### 2 The Results of the Review

To conduct the review, we followed the guidelines suggested in \[5\]. In the first step, we obtained 237 articles whose titles or abstracts contain the keywords specified in our search query. After applying the selection criteria, 38 articles were identified as \textit{primary studies} that are relevant for review. By further reviewing their related work (motivated by the fact that an article presenting a SIM most likely discusses other SIMs as related work), we identified 11 more primary studies. Among these two lists of primary studies, 19 articles are duplicates and hence resulting 30 articles as identical primary studies.

\footnote{Due to the space limitation, we do not present the review protocol that we followed. Interested readers are referred to \url{www.few.vu.nl/~qgu/SIMReviewProtocol.pdf} for details, including the specified search query, selected electronic libraries, and strategies for data extraction data synthesis.}
2.1 RQ1 What Are the Existing SIMs?

Each primary study presents one SIM and we have identified 30 SIMs altogether. An overview of the SIMs is given in Table 7. As we can see, SIM S1 and SIM S2 (hereafter labeled as S1 and S2) are the pioneering SIMs presented in 2004. The increasing number of SIMs being proposed in the last three years (6 in 2007, 9 in 2008 and 9 in 2009) reveals its increasing importance in the service engineering field.

Most of the SIMs have certain form of validation as shown in Table 7. The best way to validate a method is to put it into practice. Two primary studies describe the experience in using their methods in real life projects. Another way of validation is to experiment a SIM in case studies, which was adopted by 13 studies. In order to improve their usability, 6 primary studies provide examples in explaining how to use the proposed SIMs. For judging their quality, 4 primary studies evaluate the method in terms of e.g. survey or comparison. Only five primary studies do not discuss any validation of the proposed SIMs.

2.2 RQ2.a What Are the Different Types of Inputs?

We examined each primary study and extracted information about the input of the SIMs. We found that many SIMs start from the same type of information. For instance, legacy source code and existing software application or components are both existing software assets but in different forms. In the same vein, a collection of business models, requirements, strategies and organizational structures are all about the domain knowledge of an enterprise but describing specific enterprise elements. By comparing the inputs and classifying the ones that share the same type of information, we identified seven types of inputs. These types of inputs and the SIMs that use them are summarized in Table 1.

Table 1. Types of inputs used in the SIMs

<table>
<thead>
<tr>
<th>Type of input</th>
<th>Description</th>
<th>SIM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business process</td>
<td>A collection of related tasks or activities to fulfill a specific business requirement</td>
<td>S3, S10, S12, S15, S16, S17, S23, S24, S25, S26</td>
<td>10</td>
</tr>
<tr>
<td>Application domain</td>
<td>A collection of models or documents that describe the various aspects of an application domain, including enterprise goals or mission, business rules, business processes, business entities, organization structures, etc.</td>
<td>S1, S7, S8, S9, S14, S22</td>
<td>6</td>
</tr>
<tr>
<td>Legacy system</td>
<td>Existing software assets of an enterprise. It can be software systems, source code, or the architecture of the existing systems</td>
<td>S4, S5, S20, S27</td>
<td>4</td>
</tr>
<tr>
<td>Mix</td>
<td>A mix of type legacy system and other types</td>
<td>S2, S13, S21, S30</td>
<td>4</td>
</tr>
<tr>
<td>Data</td>
<td>The information that is processed, exchanged, or produced by business processes</td>
<td>S6, S29</td>
<td>2</td>
</tr>
<tr>
<td>Feature</td>
<td>A set of distinctive attributes or characteristics of software systems</td>
<td>S18, S19</td>
<td>2</td>
</tr>
<tr>
<td>Use case</td>
<td>A sequence of business functions that human actors benefit from</td>
<td>S11, S28</td>
<td>2</td>
</tr>
</tbody>
</table>
derived from the latter. The number of SIMs using each type of input shows that fewer methods start with more specific or technical information.

Most of the SIMs start from business processes and enterprise level information, taking both the business and its context into consideration. This is in line with the fact that service-oriented design intends to realize software re-use through large grained services especially meant to create business value.

Legacy system is another type of input and is used by four SIMs, as shown in Table 1. These SIMs take a bottom-up approach to examine the architecture (e.g., S5) or source code (e.g., S27) of the existing systems for identifying services.

As we can see, most of the SIMs are either based on domain knowledge at the enterprise level (top) or existing systems (bottom). When adopting SOA, enterprises rarely start from scratch; instead, very often they need to migrate their existing systems while creating additional services to address new business requirements. Only four SIMs take a meet-in-middle approach to identify services based on a combination of legacy systems and other information type Mix, such as domain analysis in S2 and business process models in S30. This low number is contradictory to the comparison of service analysis approaches reported in [3], which pointed out that most approaches postulate a meet-in-the-middle strategy. The cause for this contradiction lies in the fact that in our review we selected only the SIMs that provide detailed description of the methods. In [3] such a criterion does not apply and many approaches that only conceptually discuss their SI strategies have been selected for comparison. It has been admitted by the authors in [3] that many approaches do fail to go into detail. Despite of the equal importance of existing software assets and business requirements, only few SIMs provide enough guidance on how to take into account both of them.

2.3 RQ 2.b What Are the Different Types of Services Being Produced?

In this section, we discuss the outputs of the SIMs, including the types of services produced and how these services are described (output format).

Types of services. The general goal of SIMs is to identify services that are valuable, either from business or IT perspective. Each individual SIM has a specific goal in identifying specific types of services. For instance, some SIMs target at services that represent business processes whereas others focus on identifying services that bridge the gap between business services and IT infrastructure. By studying, comparing and synthesizing the data about the objectives of the services produced by the SIMs, we identified 6 types of services that have been explicitly discussed, summarized in Table 2 (note that a SIM may identify multiple types of services).

From a business perspective, we can see that 21 SIMs result in business services representing businesses processes, and 12 result in data services for business entities. Both of these two types of services are business-related. Because of the nature of services (i.e., exposing business functions), it is quite understandable that a large number of SIMs focus on business.
### Table 2. Types of outputs produced by the SIMs

<table>
<thead>
<tr>
<th>Type of output</th>
<th>Description</th>
<th>SIM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business process service (BS)</td>
<td>A service that has the business logic or represents a business process, including task services, process services.</td>
<td>S3, S7, S8, S9, S10, S11, S12, S13, S14, S15, S17, S18, S19, S21, S22, S23, S24, S25, S26, S28, S29</td>
<td>21</td>
</tr>
<tr>
<td>Data service (DS)</td>
<td>A service that represents business centric entities, including information services, entity services</td>
<td>S6, S7, S8, S9, S13, S15, S16, S17, S21, S22, S24, S25</td>
<td>12</td>
</tr>
<tr>
<td>Composite Service (CS)</td>
<td>A composition of multiple services.</td>
<td>S7, S8, S11, S12, S13, S14, S15, S17, S21, S25, S28, S29</td>
<td>12</td>
</tr>
<tr>
<td>IT service (IS)</td>
<td>A service that represents technology specific functionalities, including application services, software services, utility services and infrastructure services.</td>
<td>S3, S7, S8, S9, S10, S13, S17, S22, S24</td>
<td>9</td>
</tr>
<tr>
<td>Web service (WS)</td>
<td>A service that is implemented using the web service technology. This type is orthogonal to the other types</td>
<td>S1, S2, S4, S5, S20, S30</td>
<td>7</td>
</tr>
<tr>
<td>Partner service (PS)</td>
<td>A service that is offered to external partners.</td>
<td>S3, S16, S17, S26, S29</td>
<td>5</td>
</tr>
</tbody>
</table>

On the other hand, it is worth noticing that the numbers of SIMs for identifying IT services and partner services is relatively low and lower than we expected. Business-IT alignment is recognized as a research challenge [6] and the need of integrating technical architecture (e.g. IT infrastructure, data models) with business architecture (e.g. business models, organizational structure) has been widely agreed [12]. As shown in Table 2, all 9 SIMs that do consider IT services also identify business services and more importantly, they pay specific attention to the integration of business and IT. This alignment should, in our opinion, be supported by all the SIMs, which points out a gap in those SIMs lacking support for IT services.

As for partner services, we see that only 5 SIMs lead to services that explicitly consider their service providers (SPs) and consumers (SCs). Services are designed, developed, provided, maintained and owned by SPs. The task of SCs is to discover and compose services fulfilling their needs. SI for SPs entails how to identify services that are potentially useful to others. For SCs, it, instead, entails how to identify services for integration purposes. Because of this difference, a SIM should explicitly indicate which role it supports. Unfortunately, most of the SIMs fail to highlight this difference. Despite that the separation of SPs and SCs is considered as one of the main characteristics of SOA, these two roles are often not explicitly considered in service engineering in general, as found by a survey of SOA methodologies that we reported in [7].

Further, 7 SIMs explicitly aim at identifying web services without describing any of the types described above. That is why we regard web service as a special type, orthogonal to the others. Interestingly, nearly all of these SIMs (except for S1) rely on legacy systems.

### Types of output format

Different SIMs also describe the identified services in multiple ways and in multiple levels of detail. Some SIMs describe their output in terms of a list of services with short descriptions; whereas some others illustrate their output in terms of a model describing the relation between services and
sometimes with other artifacts of the system. To understand better the outputs of the SIMs, we analyzed the ways that the SIMs describe their identified services. As a result, five different ways of describing services have been identified, as summarized in Table 3.

### Table 3. Ways of describing outputs produced by the SIMs

<table>
<thead>
<tr>
<th>Output format</th>
<th>Description</th>
<th>SIM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informal service specification</td>
<td>Specify the identified services with a list of terms, such as service description, input, output, etc</td>
<td>S22, S7, S21, S29, S6, S9, S11, S18, S19, S25, S28</td>
<td>11</td>
</tr>
<tr>
<td>Service model</td>
<td>Model the identified services in terms of diagrams, illustrating the service landscape</td>
<td>S3, S8, S10, S12, S13, S14, S16, S24</td>
<td>8</td>
</tr>
<tr>
<td>Formal service specification</td>
<td>Describe the identified services using standard language, such as WSDL</td>
<td>S2, S15, S20, S26</td>
<td>4</td>
</tr>
<tr>
<td>Service implementation</td>
<td>Implement the identified services</td>
<td>S4, S5, S27, S30</td>
<td>4</td>
</tr>
<tr>
<td>A list of services</td>
<td>List the identified services with several key elements, such as name, operation, etc</td>
<td>S1, S17, S23</td>
<td>3</td>
</tr>
</tbody>
</table>

From Table 3, we can see that many identified services are described in informal service specification. However, different SIMs often use different terms for specifying the identified services. For instance, in S11 services are specified in detail using many terms including functional and non-function description, and technique-related aspects, such as operations and standards. In S6, however, services are specified only in terms of their operation and in/out messages. Some SIMs describe their output using only a list of services, without entering their details. While we do not enter the merit of one or the other approach, we observe that there is no unified way for describing services.

A service model is used by 8 SIMs, with the purpose of illustrating the relation between the identified services (e.g. S8) and the relation between services and their providers (e.g. S16). Thanks to its power of illustrating relations, a service model is extremely useful for describing composite services (CSs) and partner services (PSs). As shown in the overview of the SIMs (given in Table 7), only 5 (out of 12) SIMs identifying CSs, and 2 (out of 5) SIMs that identifying PSs use the form of a service model to describe the service landscape. In our opinion, a service model should be used more often as long as CSs and PSs are concerned.

Some other formats of output produced by the SIMs are implementation-driven. One of such formats is formal service specification, often used to describe services identified under more formal techniques (e.g. algorithms used by S15; another format is service implementation, often used when services as executable programs are created by wrapping source codes of legacy systems (e.g. S30).

### 2.4 R2.c What Types of Strategies and Techniques?

The previous two research questions (RQ2.a and b) mainly focus on what is involved in the SIMs. In this section we shall focus on how to carry out the SIMs. In this paper, we define strategy as the style, approach or plan for identifying services; we define technique as the technical procedures or actions taken to accomplish certain tasks defined in a SIM.
Strategies. Hubbers et. al [8] suggested ten common ways (or strategies) for identifying services. To find out if these ten ways are indeed used by the SIMs, we analyzed the data elicited from the primary studies and mapped all of the SIMs on at least one of these ways. We also identified one way (W11) that has not been discussed in [8]. All these ways and their use by SIMs are given in Table 4 (note that a SIM may use multiple strategies).

Table 4. Strategies used by the SIMs

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Description</th>
<th>SIM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1 (Business Process Decomposition)</td>
<td>Decompose business process models that depict how the work is done within an organization</td>
<td>S3, S7, S10, S12, S15, S16, S17, S21, S23, S24, S25, S26, S30</td>
<td>13</td>
</tr>
<tr>
<td>W2 (Business Functions)</td>
<td>Decompose business function models that depict what an organization does</td>
<td>S1, S2, S8, S9, S11, S18, S19, S22, S28</td>
<td>9</td>
</tr>
<tr>
<td>W3 (Business Entity Objects)</td>
<td>Model services according to business object models</td>
<td>S6, S29, S31, S16</td>
<td>4</td>
</tr>
<tr>
<td>W4 (Ownership and Responsibility)</td>
<td>Take the ownership of processes into consideration when identifying services</td>
<td>S7, S11, S28</td>
<td>3</td>
</tr>
<tr>
<td>W5 (Goal driven)</td>
<td>Decompose a company’s goals down to the level of services</td>
<td>S8, S14, S13, S22</td>
<td>4</td>
</tr>
<tr>
<td>W6 (Component-Based)</td>
<td>Identifies services based on components</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>W7 (Existing Supply)</td>
<td>Identify services from the functionality provided by existing legacy applications</td>
<td>S2, S4, S5, S13, S20, S27, S30</td>
<td>7</td>
</tr>
<tr>
<td>W8 (Front-Office Application Usage Analysis)</td>
<td>Select a set of applications that support business processes and extracts comparable functions into a single service</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>W9 (Infrastructure)</td>
<td>Keep the technical infrastructure into consideration when identifying services</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>W10 (NFRs)</td>
<td>Use non-functional requirements as the primary input to identify services</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>W11 (User interface)</td>
<td>Identify services based on the design of user interface</td>
<td>S15</td>
<td>1</td>
</tr>
</tbody>
</table>

Four strategies (W6, 8, 9, 10) are discussed in [8] but have not been used by any of the SIMs that we studied. As explained in [8], W6 has practical difficulties due to the different nature of component and services; W8 might be risky if existing application design is of low quality; and W9 results in services heavily coupled with infrastructure. Due to these issues, it is understandable that the SIMs avoid using these strategies. W10 points out the importance of non-functional requirements (NFRs) in SI as conflicting NFRs might cause redesign. As no SIM relies on W10, further research is required in supporting NFRs.

Interestingly, we also identified a new strategy, W11 user interface (not discussed in [8]). User interface (UI) design, an integral part of the software design process, is often considered out of scope of SOA design [9]. However, a UI design helps one to distinguish the purely presentation aspects from data and process aspects and hence aids the identification of services. The use of UI design in SI is regarded as an innovative approach.

Some SIMs use only one strategy (W1, 2, 3 or 7) and we call these strategies primary strategies; while the others are always used in combination with the primary strategies and we call them complementary strategies. The most popular primary strategies are W1, 2 and 7, used by 13, 9 and 7 SIMs respectively. The first two are top-down approaches by examining the business requirements while
the last one represents a bottom-up approach by extracting valuable and/or reusable functions from existing applications. In most of the cases, the strategies used by a SIM are directly related to its input. E.g., all the SIMs that use business process as their input use \( W_1 \) (decompose business process) as its strategy, only S15 exceptionally uses a combination of \( W_4 \) and \( W_{11} \).

The complementary strategies are \( W_4, 5 \) and \( W_{11} \) which are specifically aided by the information about goals, stakeholders and user interfaces. However, this information alone is often not sufficient for identifying services. For instance, in S15 user interfaces are first designed based on business process decomposition and then services are identified by extracting abstract WSDL specifications from user interface specifications. Obviously using \( W_{11} \) (user interface) only is not sufficient in this example. The use of these complementary strategies often results in services more business-driven as explained in e.g. [10].

**Techniques.** After studying the data about the techniques used in the SIMs, we have identified six different types of techniques, summarized in Table 5.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Description</th>
<th>SIM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
<td>A formal approach to problem solving, such as heuristic or formalized rules</td>
<td>S1, S2, S3, S5, S8, S11,</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S15, S16, S17, S25, S26</td>
<td></td>
</tr>
<tr>
<td>Guidelines</td>
<td>A set of pre-defined regulations, such as criteria or policies; suggested but not codified</td>
<td>S7, S9, S10, S14, S28, S29, S30</td>
<td>7</td>
</tr>
<tr>
<td>Analysis</td>
<td>A process of studying, interpreting, reasoning and synthesizing data</td>
<td>S4, S12, S19, S18, S21, S22</td>
<td>6</td>
</tr>
<tr>
<td>Ontology</td>
<td>A technique to conceptually represent (domain) knowledge</td>
<td>S24, S23, S19, S27</td>
<td>4</td>
</tr>
<tr>
<td>Pattern</td>
<td>Defined as recurring solution to recurring problems</td>
<td>S6</td>
<td>1</td>
</tr>
<tr>
<td>Information</td>
<td>A text process techniques for identifying or eliciting useful information, such as information retrieval or textural similarity analysis</td>
<td>S20</td>
<td>1</td>
</tr>
</tbody>
</table>

Some of them are more formal, in the sense that they formally codify formulas or rules to specify the way that services are identified, such as algorithm, ontology, pattern and information manipulation. Nearly half of the SIMs use these formal techniques and thanks to the advantage of codification, some SIMs partially automate the SI process. For instance, in S17 a tool called SQUID was developed to automate the process of SI and in S2 executable algorithms are used to analyze the legacy code.

A less formal technique is guidelines, which is used by 7 SIMs as shown in Table 5. These SIMs provide advices like how to identify the right-grained services from goal-scenario models (S14) and how to map tasks in business process models to services (S30).

Different from these relatively formal techniques, analysis is a technique that is more abstract and requires its users to deeply understand the problem they face and make motivated decisions. Accordingly, the subjectivity of using the technique is relatively high and different actors may achieve different results by applying the same SIM.
3 An Input-Output Matrix for the Selection of SIMs

The variety in the types of inputs, outputs and processes discussed in Sec. 2 explains why practitioners often face difficulties to select a SIM that both fits their needs and copes with the available resources. To help compare and select among the SIMs, we use these results (summarized in Table 7) to created an input-output matrix. The matrix is presented in Table 6, where rows represent the types of outputs produced by the SIMs and columns represent the types of inputs being used. Each cell of the matrix describes a specific SIM (in terms of its output format, strategy and technique) if it uses the input and produces the output represented by the column and row respectively. For instance, a SIM that uses Application domain as its input (column 1) and produces Business services as well as composite service (BS+CS) (row 2) is S14, whose output is described in terms of a service model (SM), uses goal driven (W5) as its strategy and guidelines as technique. In the following, we shall explain how this matrix aids the selection of SIMs in three different ways.

Selection driven by the targeted output. Sometimes, before performing the task of SI, it is expected that certain types of services are identified. For instance, an enterprise that focuses on improving the efficiency and maintainability of its business processes may be interested in business services; while an enterprise that intends to expose its business functions to other partners for business collaboration might be also interested in partner services. In our input-output matrix, the SIMs are classified horizontally, according to the types of services they produce. When the target types of services to be identified are decided, one can use the matrix to eliminate those SIMs that are irrelevant to the needs. Suppose partner services (PS) are of great importance, one can see from our matrix that five SIMs (row 8 to 12) could be selected. By determining more types of services to be identified (e.g. DS), one can further narrow down the number of candidate SIMs (e.g. S17 and S16 at row 10 and 12). Further, the matrix also provides a straightforward view on what types of inputs are needed if certain SIMs are selected. For example, we can see from the matrix that to identify partner services, either business processes or business centric entities (data) should be known. This helps one to judge the feasibility of the SIMs.

Selection driven by the available resources. Knowing what kinds of resources are available for SI, one can also choose a SIM based on its starting point. In the input-output matrix, the SIMs are classified vertically, according to the types of inputs they start with. Using the matrix, one can have an overview of what types of services can be produced given the available resources and at the same time find out the SIMs that support these resources. For instance, if the only available resource is a set of use cases (see Table 6 column 7) describing some business functions, one can find from our matrix that two SIMs, S11 and S28 (column 7, row 2) start SI from this resource. Accordingly, one can also expect that business services and their compositions (BS+CS) can be identified using either of these two SIMs.
### Table 6. Input-output matrix of the SIMs

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BS</td>
<td></td>
<td>S23(List, W1, Onto)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS+CS</td>
<td></td>
<td>S14(SM, W5, Gline)</td>
<td>S12(SM, W1, Anal)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS+DS+CS</td>
<td></td>
<td>S15(FSP, W1&amp;11, Algo)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS+DS+IS+CS</td>
<td></td>
<td>S9(ISP, W2, Gline)</td>
<td>S24(SM, W1, Onto)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BS+IS</td>
<td></td>
<td>S10(SM, W1, Gline)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DS</td>
<td></td>
<td>S6(ISP, W3, Patt)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS+BS</td>
<td></td>
<td>S26(FSP, W1, Algo)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS+BS+CS</td>
<td></td>
<td>S17(List, W1, Algo)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS+BS+DS+CS</td>
<td></td>
<td>S3(SM, W1, Algo)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PS+DS</td>
<td></td>
<td>S16(SM, W1&amp;3, Algo)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legenda: **Output format**: List - A list of services, ISP: Informal service specification, FSP - Formal service specification, SM - Service model, SI - Service implementation; **Technique**: Algo - Algorithm, Gline - Guideline, Onto - Ontology, Anal - Analysis, Patt - Pattern, Info - Information manipulation

### Selection by comparison of alternative SIMs

Some SIMs can be seen as alternative methods when they share the same type of inputs and outputs. Despite of these commonalities, these methods often differ in the way that the outputs are described, and/or the strategy and technique they use. Using the input-output matrix, one can easily pinpoint and compare these alternative methods since they are grouped and located in the same cell of the matrix. For instance, given *legacy system* (column 3) as starting point and *web services* (row 13) as output, the matrix shows four SIMs: S4, S5, S20 and S27. Comparing these four SIMs in terms of their output formats, strategies and techniques, we can see that the main difference lies in the techniques they use. Therefore, one can choose among these four SIMs based on their preference of one technique over another, depending available technologies, competencies in place, etc. To give another
example, given application domain (column 1) as starting point and BS+DS+IS (row 4) as output, our matrix shows two SIMs: S9 and S22. By comparing these two SIMs, it is also easy to see that S22 uses W5 (goal driven) complementary to W2 (business functions) as its strategy; and thereby an enterprise that have clearly defined business goals might opt to select S22 over S9. However, if the enterprise prefers to follow guidelines to provide relatively objective results than to perform analysis to produce relatively subjective results, it might select S9 over S22. As such, our matrix provides a way to preliminarily select alternative SIMs before more comprehensive comparison (if needed).

4 Conclusion

In this paper, we report the classification and comparison of 30 SIMs identified from a systematic literature review. The many different types of inputs, outputs and processes of the SIMs show a significant heterogeneity. Our results provide a holistic overview of the SIMs and highlight their differences. To help practitioners compare and select from these SIMs, we created an input-output matrix that aids the selection of SIMs in three different but complementary ways.

Further, the findings of this review outlines future research directions to further improve the existing SIMs and to guide the design of new SIMs. Our main observations are 1) IT services that leverage business processes and underlying IT infrastructure require more attention (Sec. 2.3); 2) Services for internal use and external consumption should be differentiated due to their different characteristics (Sec. 2.3); and 3) NFRs should be explicitly considered due to their importance through the entire service life cycle (Sec. 2.4).

References


Table 7. An overview of the existing SIMs (Appendix)

<table>
<thead>
<tr>
<th>SIM</th>
<th>Year</th>
<th>Type of input</th>
<th>Strategy</th>
<th>Output format</th>
<th>Type of output</th>
<th>Technique</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>2004</td>
<td>Application domain</td>
<td>W2</td>
<td>WS</td>
<td>A list of services</td>
<td>Algorithm</td>
<td>Evaluated</td>
</tr>
<tr>
<td>S2</td>
<td>2004</td>
<td>Mix</td>
<td>W2&amp;7</td>
<td>WS</td>
<td>Formal service specification</td>
<td>Algorithm</td>
<td>Evaluated</td>
</tr>
<tr>
<td>S3</td>
<td>2005</td>
<td>Business process</td>
<td>W1</td>
<td>PS+BS+IS</td>
<td>Service model</td>
<td>Algorithm</td>
<td>No</td>
</tr>
<tr>
<td>S4</td>
<td>2005</td>
<td>Legacy system</td>
<td>W7</td>
<td>WS</td>
<td>Service implementation</td>
<td>Analysis</td>
<td>Case study</td>
</tr>
<tr>
<td>S5</td>
<td>2005</td>
<td>Legacy system</td>
<td>W7</td>
<td>WS</td>
<td>Service implementation</td>
<td>Algorithm</td>
<td>Case study</td>
</tr>
<tr>
<td>S6</td>
<td>2006</td>
<td>Data</td>
<td>W3</td>
<td>DS</td>
<td>Informal service specification</td>
<td>Pattern</td>
<td>No</td>
</tr>
<tr>
<td>S7</td>
<td>2007</td>
<td>Application domain</td>
<td>W1&amp;4</td>
<td>BS+DS+IS-+CS</td>
<td>Informal service specification</td>
<td>Guidelines</td>
<td>Case study</td>
</tr>
<tr>
<td>S8</td>
<td>2007</td>
<td>Application domain</td>
<td>W2&amp;5</td>
<td>BS+DS+IS-+CS</td>
<td>Service model</td>
<td>Algorithm</td>
<td>No</td>
</tr>
<tr>
<td>S9</td>
<td>2007</td>
<td>Application domain</td>
<td>W2</td>
<td>BS+DS+IS</td>
<td>Informal service specification</td>
<td>Guidelines</td>
<td>Project</td>
</tr>
<tr>
<td>S10</td>
<td>2007</td>
<td>Business process</td>
<td>W1</td>
<td>BS+IS</td>
<td>Service model</td>
<td>Guidelines</td>
<td>No</td>
</tr>
<tr>
<td>S11</td>
<td>2007</td>
<td>Use case</td>
<td>W2&amp;4</td>
<td>BS+CS</td>
<td>Informal service specification</td>
<td>Algorithm</td>
<td>Example</td>
</tr>
<tr>
<td>S12</td>
<td>2007</td>
<td>Business process</td>
<td>W1</td>
<td>BS+CS</td>
<td>Service model</td>
<td>Analysis</td>
<td>Example</td>
</tr>
<tr>
<td>S13</td>
<td>2008</td>
<td>Mix</td>
<td>W5&amp;7</td>
<td>BS+DS+IS-+CS</td>
<td>Service model</td>
<td>Analysis</td>
<td>Case study</td>
</tr>
<tr>
<td>S14</td>
<td>2008</td>
<td>Application domain</td>
<td>W5</td>
<td>BS+CS</td>
<td>Service model</td>
<td>guidelines</td>
<td>Case study</td>
</tr>
<tr>
<td>S15</td>
<td>2008</td>
<td>Business process</td>
<td>W1&amp;11</td>
<td>BS+DS+CS</td>
<td>Service implementation</td>
<td>Algorithm</td>
<td>Case study</td>
</tr>
<tr>
<td>S16</td>
<td>2008</td>
<td>Business process</td>
<td>W1&amp;3</td>
<td>PS+DS</td>
<td>Service model</td>
<td>Algorithm</td>
<td>Evaluated</td>
</tr>
<tr>
<td>S17</td>
<td>2008</td>
<td>Business process</td>
<td>W1</td>
<td>PS+BS+DS++IS+CS</td>
<td>A list of services</td>
<td>Algorithm</td>
<td>Example</td>
</tr>
<tr>
<td>S18</td>
<td>2008</td>
<td>Feature</td>
<td>W2</td>
<td>BS</td>
<td>Informal service specification</td>
<td>Analysis</td>
<td>Case study</td>
</tr>
<tr>
<td>S19</td>
<td>2008</td>
<td>Feature</td>
<td>W2</td>
<td>BS</td>
<td>Informal service specification</td>
<td>Ontology</td>
<td>Case study</td>
</tr>
<tr>
<td>S20</td>
<td>2008</td>
<td>Legacy system</td>
<td>W7</td>
<td>WS</td>
<td>Service implementation</td>
<td>Information manipulation</td>
<td>Case study</td>
</tr>
<tr>
<td>S21</td>
<td>2008</td>
<td>Mix</td>
<td>W1&amp;3</td>
<td>BS+DS+CS</td>
<td>Informal service specification</td>
<td>Analysis</td>
<td>Example</td>
</tr>
<tr>
<td>S22</td>
<td>2009</td>
<td>Application domain</td>
<td>W2&amp;5</td>
<td>BS+DS+IS</td>
<td>Informal service specification</td>
<td>Analysis</td>
<td>No</td>
</tr>
<tr>
<td>S23</td>
<td>2009</td>
<td>Business process</td>
<td>W1</td>
<td>BS</td>
<td>A list of services</td>
<td>Ontology</td>
<td>Case study</td>
</tr>
<tr>
<td>S24</td>
<td>2009</td>
<td>Business process</td>
<td>W1</td>
<td>BS+DS+IS</td>
<td>Service model</td>
<td>Ontology</td>
<td>Case study</td>
</tr>
<tr>
<td>S25</td>
<td>2009</td>
<td>Business process</td>
<td>W1</td>
<td>BS+DS+CS</td>
<td>Informal service specification</td>
<td>Algorithm</td>
<td>Case study</td>
</tr>
<tr>
<td>S26</td>
<td>2009</td>
<td>Business process</td>
<td>W1</td>
<td>PS+BS</td>
<td>Service implementation</td>
<td>Algorithm</td>
<td>Evaluated</td>
</tr>
<tr>
<td>S27</td>
<td>2009</td>
<td>Legacy system</td>
<td>W7</td>
<td>WS</td>
<td>Service implementation</td>
<td>Ontology</td>
<td>Case study</td>
</tr>
<tr>
<td>S28</td>
<td>2009</td>
<td>Use case</td>
<td>W2&amp;4</td>
<td>BS+CS</td>
<td>Informal service specification</td>
<td>Guidelines</td>
<td>Example</td>
</tr>
<tr>
<td>S29</td>
<td>2009</td>
<td>Data</td>
<td>W3</td>
<td>PS+BS+CS</td>
<td>Informal service specification</td>
<td>Guidelines</td>
<td>Example</td>
</tr>
<tr>
<td>S30</td>
<td>2009</td>
<td>Mix</td>
<td>W1&amp;7</td>
<td>WS</td>
<td>Service implementation</td>
<td>Guidelines</td>
<td>Case study</td>
</tr>
</tbody>
</table>