FACING UNCERTAINTY IN WEB SERVICE COMPOSITIONS

1Germán H. Alférez, 2Vicente Pelechano
1Facultad de Ingeniería y Tecnología, Universidad de Montemorelos, Mexico, 2Centro de Investigación en Métodos de Producción de Software (ProS), Universitat Politècnica de València, Spain

1harveyalferez@um.edu.mx, 2pele@dsic.upv.es

Abstract

Web service compositions run in complex computing infrastructures where arising events may affect the quality of the system. However, crucial Web service compositions cannot be stopped to apply changes to deal with problematic events. Therefore, the trend is moving towards context-aware Web service compositions, which use context information as a basis for autonomic changes. Under the closed-world assumption, the context and possible adaptations are fully known at design time. Nevertheless, it is difficult to foresee all the possible situations arising in uncertain contexts. In this article, we leverage models at runtime to guide the dynamic evolution of context-aware Web service compositions to deal with unexpected events in the open world. In order to manage uncertainty, a model that abstracts the Web service composition, self-evolves to preserve requirements. The evolved model guides changes in the underlying WS-BPEL composition schema. A software process model and its supporting method content are provided to guide the construction of models and other artifacts at design time. A prototype and an evaluation demonstrate the feasibility of our approach.

Keywords: Uncertainty, Web service compositions, models at runtime, dynamic evolution, open world, process model, method content.

1. INTRODUCTION

Today's systems run in complex and heterogeneous computing infrastructures in which a diversity of events may arise (e.g. security threats and server failures). Therefore, it is desirable to count on self-adjusting mechanisms to solve these situations. A good example of systems that require adjusting themselves are the ones based on Web service compositions (hereafter, service compositions). In an ideal scenario, Web service operations would do their job smoothly. However, several exceptional situations may arise in the changing contexts where they run. For example, the response time of a Web service operation may have greatly increased. Therefore, it is appropriate to count on context-aware service compositions that dynamically change to keep service-level agreements (SLAs), offer extra functionality, protect the system, or make the system more usable. Dynamic adaptation refers to the act of changing the software behavior as it executes, without stopping it. This type of adaptation is especially important in critical service compositions that cannot be stopped to implement adaptations.

Although current research works have paved the way towards the dynamic adaptation of service compositions, they have two main drawbacks: 1) most solutions have tended to implement dynamic adaptations with variability constructs at the language level (Colombo, Di Nitto, & Mauri, 2006; Koning, Sun, Sinnema, & Avgeriou, 2009; Baresi & Guinea, 2011; Sonntag & Karastoyanova, 2011) (e.g. by extending WS-BPEL code). However, this approach makes it harder to understand and communicate autonomic behavior decisions among stakeholders. This may result in error-prone systems. Moreover, any change in the platform or the application may require to change low-level platform-specific scripts that are tightly coupled with the application code. This process may become tedious and time-consuming; 2) dynamic adaptations are carried out in the closed world, in which the boundary between system and context is known ahead and unchanging (Baresi, Di Nitto, & Ghezzi, 2006). In this scenario, a set of adaptation actions is predefined for fully foreseen context events (Bosloper, Siljée, Nijhuis, & Hammer, 2005; Menasce, Gomaa, Malek, & Sousa, 2011; Alférez & Pelechano, 2011). However, in the unpredictable open world, service compositions should react to continuous and unanticipated changes in the context.

In the open world, uncertainty is caused by how the service composition should deal with unknown context events. Unknown context events are those situations in the context that have not been foreseen at design time (Alférez & Pelechano, 2012).

In this article, we try to reduce uncertainty in the open world by self-evolving service compositions through models at runtime. Models at runtime are causally connected self-representations of the associated system that emphasize the structure, behavior, or goals of the system from a problem space perspective (Blair, Bencomo, & France, 2009). At the time unknown context events arise during execution, our approach triggers the dynamic evolution of the service composition to manage these events properly. To this end, easy-to-understand and technology-independent models are used to describe self-adjusting actions that preserve the...
expected requirements. The evolved models guide changes in the underlying WS-BPEL composition schema, which orchestrates the service composition. As a result, dynamic evolutions move the service composition to new versions, which cannot be supported by predefined dynamic adaptations. Since we are interested in managing uncertainty that arises from the context in which the service composition is deployed, our approach is related to external uncertainty (Esfahani, Kouroshfar, & Malek, 2011).

The remainder of this article is structured as follows: Section 2 describes a running example. Section 3 summarizes the dimensions of dynamic adaptation and dynamic evolution in service compositions. Section 4 presents an overview of our approach for dynamic evolution of service compositions. Section 5 describes a process model and its supporting method content to guide the construction of the models and other artifacts to guide dynamic evolutions. Section 6 summarizes the mechanisms that support evolution. Section 7 describes a prototype that realizes our approach and evaluation results. Section 8 summarizes the mechanisms that support evolution. Section 9 presents conclusions and future work.

2. Running Example

In order to illustrate the need for dealing with uncertainty in the open world, we introduce a critical service composition for online book shopping. The example is specified with the Business Process Model and Notation (BPMN) in Figure 1. BPMN tasks express Web service operations (e.g. UPS Shipping); and BPMN subprocesses express composite service operations (e.g. Barnes & Noble Books).

The business process (BP) starts when a customer looks for a book on the store's website. The first thing the customer wants to do is to identify the books to purchase. The searching operation is provided by the Search Book Web service, which is part of the Barnes & Noble Books composite service. When a book is found, then the book information is returned to the customer by the Show Book Info Web service while at the same time the information for other related books is listed by the Show Related Titles Web service. If no book is found, then the customer must refine the search, e.g. using supplementary or different search criteria, or undertaking another search. In the next step, the customer adds books into the shopping cart through the Barnes & Noble Shopping Cart Web service. The process can start over again until the customer is satisfied with his or her selection. When the customer is ready to checkout, he or she has to be authenticated by the Google Authentication Web service. Then, the in-house Payment Calculator Web service calculates the total amount to be paid. The payment is done through the Bank of America Credit Card Payment Web service. Finally, if the credit card information is valid, the in-house E-mail Invoice Web service sends an e-mail to the customer with the invoice while the UPS Shipping Web service is invoked to deliver the book. Otherwise, the process terminates.

A set of adaptation actions have been created for unforeseen context events. For instance, if the Barnes & Noble Books composite service operation is unavailable, then other service operations can be invoked instead. Nevertheless, if there are no predefined adaptation actions for unknown context situations (e.g. any third-party Web service operation fails or performs below required SLAs), then no adaptation is carried out. As a result, the whole system quality may be harmed. We argue that despite unknown context events, the service composition has to keep offering expected requirements.

3. Dimensions of Dynamic Adaptation and Dynamic Evolution in Service Compositions

In order to introduce our approach, first it is necessary to
describe the dimensions of dynamic adaptation and dynamic evolution in service compositions. Figure 2 summarizes these dimensions. The two dimensions in the lower section are about what should be changed according to information retrieved from the context. In the dimension of dynamic adaptation of running instances, only the instance that has triggered the adaptation is adapted. In contrast, the dimension of dynamic evolution of the composition schema modifies the composition schema (e.g. described in WS-BPEL), and therefore all future instances. These two dimensions have been widely described in literature (Marconi et al., 2009; Weber, Reichert, & Rinderle-Ma, 2008).

On the other hand, the two dimensions in the top of Figure 2 are related to the context of execution. The dimension of dynamic adaptation in the closed world is focused on the closed-world assumption, in which all context events are foreseen at design time. Predefined actions guide adaptations according to known context events (for example, through event-condition-action – ECA – rules). There are several research works that focus on this dimension (Bosloper et al., 2005; Menasce et al., 2011; Alférez & Pelechano, 2011). However, predefined adaptation actions for known context events are not enough in the open world where several unknown context events can arise (e.g. sudden security attacks). Despite the recognized need for handling unexpected events in self-adaptive systems (Calinescu, Ghezzi, Kwiatkowska, & Mirandola, 2012; Cheng, Lemos, Giese, Inverardi, & Magee, 2009), the dimension of dynamic evolution in the open world of service compositions is still an open and challenging research topic. Our work focuses on this dimension.

![Figure 2. Dimensions of Dynamic Adaptation and Dynamic Evolution in Service Compositions](http://hipore.com/ijsc)

### 4. Facing Uncertainty with Dynamic Evolution

In our previous work (Alférez & Pelechano, 2011), we described a model-driven approach to support the dimension of dynamic adaptation of service compositions in the closed world. In order to support the dynamic evolution of service compositions in the open world, we extend this dimension with a dynamic evolution layer. Highly abstract tactics are the main components to face uncertainty in service compositions that run in the open world. The use of tactics is common in sports, war, or even in daily matters to accomplish an end. For example, the most important goal during a battle is to win. However, unknown or unforeseen events, such as surprise assaults, may arise. These events may negatively affect the expected goal. Therefore, it is necessary to choose among a set of tactics to reach the goal (e.g. to escape vs. to do a frontal attack).

We define tactics as last resort surviving actions or strategies to preserve the requirements (i.e., goals) that can be negatively impacted by unknown context events (Alférez & Pelechano, 2012). Therefore, tactics can trigger the dynamic evolution of a service composition to preserve its requirements, which were defined at design time.

There is a main difference between tactics and compensations mechanisms. On one hand, compensation mechanisms try to reverse actions performed in a transaction when failures are faced. On the other hand, tactics try to preserve expected requirements that may be affected by arising problematic unknown context events.

During dynamic evolutions, requirements and tactics need to be known beforehand. Otherwise, it will be impossible to face uncertainty in a controlled way. However, they are not attached to any context event or specific reconfiguration actions (as dynamic adaptations do (Alférez & Pelechano, 2011)). Therefore, our approach manages known unknowns: tactics are known beforehand, but we do not know to which specific arising unknown context events they will be applied.

In the open world, our approach tries to reduce the impact of unknown context events with a group of tactics. Therefore, the open world can be seen as:

\[
\text{Open world} = (\sum \text{unknown context events that can be handled by tactics}) \cup (\sum \text{unhandled unknown context events}).
\]

Figure 3 shows the architecture of the dynamic evolution layer to face uncertainty in the open world. This architecture has three building blocks: 1) the Evolution Planner constantly looks for unknown context events in the open world. If there is an unknown context event, then it looks for a requirement that can be affected by this event. Afterwards, it looks for a surviving tactic to preserve the requirement; 2) the Reconfiguration Engine performs the necessary evolution in two main steps. First, it merges the discovered tactic into a composition model, which abstracts the service composition. This evolved composition model supports the tactic’s functionality to preserve the affected requirement. Second, it evolves the WS-BPEL composition schema according to the evolved composition model; and 3) the Execution Engine deploys the evolved WS-BPEL composition schema.
5. SUPPORTING MODELS AND OTHER ARTIFACTS FOR DYNAMIC EVOLUTION

Our approach requires pieces of knowledge during execution to reach dynamic evolution of service compositions. These pieces are defined as abstract models and other supporting artifacts. In order to facilitate the comprehension and applicability of the tools that are required to create these models and artifacts, it is necessary to describe them in the context of a software process model. As a result, it will be possible to answer the following question: **Which activity of the software process can be supported by particular tools?** In addition, it is necessary to describe who uses the tools, what are the required inputs, and what are the resulting outputs.

A well-known modeling language to describe software processes is the Software & Systems Process Engineering Metamodel (SPEM). Specifically, SPEM 2.0 is the Object Management Group (OMG) standard for defining software and systems development processes and their components. The conceptual SPEM framework has two key elements:

1. **Method content:** It contains libraries of reusable elements, such as roles, work products, and tools. It defines the **who, what, and how** of work increments that have to be done.
2. **Process:** It defines the **when** of work increments that have to be done. The creation of processes is supported by reusable method contents.

Figure 4 shows the proposed process model to design the dynamic evolution of service compositions. This process model is composed of a sequence of activities to be followed.

In the process model of Figure 4, the composition model is created first. This model abstracts the underlying service composition. Then, the requirements model is created to count on an abstraction that expresses the needs and constraints placed on a service composition. Afterwards, tactic models are created in order to preserve the expected requirements during execution. Tactic models are expressed as composition models. This level of abstraction is necessary in order to express the workflow among service operations in tactics. Then, we propose to 1) implement the service operations (atomic and composite service operations) for tactics; and 2) the fragments of WS-BPEL code to invoke these service operations. Finally, we propose to create the rule premises to find the requirements that can be affected by unknown context events.

In order to guide the creation of the proposed process model, we implemented a plugable SPEM-based method content. The main benefit of this plugin is that other process models can reference them and reuse their content. The EPF Composer\(^1\) was used to create this plugin.

---

\(^1\) EPF Composer: [http://www.eclipse.org/epf](http://www.eclipse.org/epf)
The following subsections present the tasks that support the creation of the work products to guide dynamic evolutions. Each task has a one-to-one mapping to each activity in the process model depicted in Figure 4. Therefore, the information in each task supports the creation of each activity in the process model. The description of each task contains the information about the tools that are used to create work products.

5.1 Create the Composition Model

The objective of this task is to create a composition model that abstracts the service composition (see Figure 5). The composition model abstracts the underlying service composition (e.g. the model in Figure 1).

A BPMN model was chosen to represent the elements in a service composition because BPMN is suitable to express sequences and dependencies among Web services and composite services.

The composition model can be created with the STP BPMN Modeler\(^2\) or with the infrastructure provided by the Eclipse Modeling Framework (EMF)\(^3\). Also, this model can be created proactively (before the service composition is programmed) or reactively (based on a running service composition). In case of choosing the proactive way, we propose a model-driven process to facilitate the creation of the WS-BPEL composition schema. This process is depicted in Figure 6. The Babel\(^4\) project can be used to translate the BPMN composition schema into WS-BPEL code. Babel is an open-source tool that integrates a set of techniques to translate models captured using a core subset of BPMN into WS-BPEL.

Before performing the Babel to Initial WS-BPEL activity, the composition model must be prepared according to the format accepted by the Babel tool. This is performed in the BPMN to BP-Babel activity by means of a model-to-model transformation. Although the Babel tool uses model-to-text transformations to generate a WS-BPEL document from a BPMN model (in the Babel to Initial WS-BPEL activity), the generated WS-BPEL document is incomplete. For example, it lacks information about the partner links of services participating in the process and the variables used in the process. In order to solve this situation, we created a graphical tool based on

\(^2\) BPMN Modeler: http://git.eclipse.org/c/bpmnmodeler

\(^3\) EMF: http://www.eclipse.org/emf/

Forms Modeling Framework (FMF)\(^5\) that allows systems analysts input the missing information in the WS-BPEL document generated with Babel. FMF is a part of the Modeling Software Kit (MOSKitt) framework\(^6\).

Thanks to our FMF-based configuration tool, the WS-BPEL Completion activity is carried out as follows. A video demonstration of this tool is available online\(^7\):

1. Our FMF-based configuration tool looks for patterns in the Babel’s generated WS-BPEL code in order to fill up an intermediate model that can be handled by FMF. The intermediate model conforms to the WS-BPEL metamodel. Our tool uses the JDOM\(^8\) libraries and XML Path Language (XPATH)\(^9\) in order to do the following: 1) manipulate the WS-BPEL composition schema (a XML file); and 2) look for WS-BPEL elements such as invokes, imports, whiles, switches, variables, and namespaces to be completed.

2. The systems analyst inputs the missing information in our tool in order to complete the WS-BPEL composition schema. Specifically, the following information is introduced in FMF forms: 1) information about Web services, such as partner links, port types, input and output variables, and necessary WSDLs; 2) information about assign elements that manipulate data; 3) information to complete the behavior of whiles, switches, and timers.

   Figure 7 shows the graphical interface of this tool. The pattern that is used in most of the forms is the following. Section A shows a list with all the elements of a WS-BPEL element group (e.g., partner links). Section B shows a set of tabs to have access to the forms that allow to add information to elements in particular WS-BPEL element groups. Section C shows the attributes associated to a particular selected element in Section A. These attributes can be modified by the systems analyst. Since a WS-BPEL composition schema has only one Process element, its related form omits Section C.

3. Once the information has been completed in the intermediate model, our tool uses this information to automatically complete the WS-BPEL composition schema. The steps that are carried out in this operation are as follows: 1) the tool looks for each element that has been completed in the intermediate model; 2) the tool looks in the WS-BPEL file generated by Babel for the elements that have been completed in the intermediate model; and 3) the tool completes the missing information in the WS-BPEL file generated by Babel with the information in the intermediate model. Our tool can also use the completed intermediate model to automatically generate other artifacts to execute the process in the Execution Engine, such as the WSDL file to access the service composition and

---

\(^5\) FMF-Based Tool to complete WS-BPEL schemata: http://download.moskitt.org/moskitt/fmf/updates-1.3.8/

\(^6\) MOSKitt: http://www.moskitt.org

\(^7\) Video demonstration of our FMF-Based tool: https://vimeo.com/58989214

\(^8\) JDOM: http://www.jdom.org

\(^9\) XPATH: http://www.w3.org/TR/xpath

http://hipore.com/ijsc
the deployment descriptor (e.g. the deploy.xml file in Apache ODE\textsuperscript{10}).

![Figure 8. Create the Requirements Model Task](image)

### 5.2 Create the Requirements Model

The objective of this task is to create a requirements model, which abstracts the requirements that the service composition has to preserve at runtime (see Figure 8). The requirements engineer role is in charge of creating the requirements model work product.

In our approach, we are particularly interested in keeping non-functional requirements (NFRs) at runtime. Therefore, we propose to use the Goal-oriented Requirements Language (GRL) (Liu & Yu, 2004) for requirements modeling because it is focused on NFRs. Figure 9 shows the requirements model for our running example. Softgoals describe the NFRs to be kept by the service composition in order to reach the top-level goal, and tasks specify particular surviving tactics to reach softgoals. There is a one-to-many relationship between a requirement (expressed as a softgoal in our case) and tactics that can preserve it. A large set of available generic tactics can be found in related work (Bass, Clements, & Kazman, 2012).

Conflicts may exist between tactics and softgoals. For example, the application of the Deception tactic that preserves the High Security softgoal can negatively impact the High Performance softgoal (more computational resources are required). In order to solve conflicts, we define claims, which indicate assumptions about operationalizations’ satisfaction of softgoals (Sawyer, Mazo, Diaz, Salinesi, & Hughes, 2012). Tactics specified in claims use an ordinal scale, ranging from complete denial (−−) to complete satisfaction (++++) to express softgoals satisfaction. If a claim states that a tactic operationalization has a negative impact on a softgoal, then another tactic can be tried.

For example, Figure 9 shows two claims. At runtime, if an unknown context event affects the High Security softgoal, the Evolution Planner chooses the Deception tactic first according to C1 because it has a better impact on this softgoal. If this tactic does not solve the problem, then the Temporal Separation tactic can be triggered. In a more complex scenario, if an unknown context event affects the High Security and High Performance softgoals at the same time, C2 is checked to decide the best tactic to choose from. In this case, the Temporal Separation and Introduce Concurrency are chosen because they have the most positive impact on these softgoals.

We choose OpenOME\textsuperscript{11} to create the requirements model because it offers an open metamodel that can be used to create XML Metadata Interchange (XMI)-based requirements models. OpenOME is an open source, general, goal-oriented and/or agent-oriented modeling and analysis tool. It provides users with a graphical interface to develop models, and supports access to a knowledge base that allows for sophisticated computer-aided analysis. OpenOME supports the $i^*$ framework.

In our approach, we manage four kinds of OPENOME contributions: 1) the MakeContribution is a positive contribution strong enough to satisfice a softgoal (i.e., ++); 2) the HelpContribution is a partial positive contribution, not sufficient by itself to satisfy the softgoal (i.e., +); 3) the BreakContribution is a negative

![Figure 9. Fragment of the Requirements Model for the Running Example](image)

\textsuperscript{10} Apache ODE: http://ode.apache.org
\textsuperscript{11} OpenOME: https://se.cs.toronto.edu/trac/ome

http://hipore.com/ijscc
contribution sufficient enough to deny a softgoal (i.e., \(-\)-); and 4) the \textit{HurtContribution} is a partial negative contribution, not sufficient by itself to deny the softgoal (i.e., \(-\)).

Although the OPENOME metamodel contains the \textit{Belief} metaclass, its graphical interface misses the representation for claims (or beliefs). Therefore, it is necessary to use the EMF generic editor in order to add claims into the requirements model.

5.3 Create the Tactic Models

The objective of this task is to create a set of tactic models (see Figure 10). Tactic models express the tactical functionality to be triggered on the service composition to preserve affected requirements. Therefore, tactic models are causally connected to software (e.g. Web services) that implements the tactics. Tactic models are merged into the composition model at runtime to include the tactical functionality in the evolved service composition. The only merging prerequisite is that these two models conform to the same metamodel.

Figure 11 shows a tactic model in our running example. Since the composition model is implemented as a BPMN model, this tactic model is also expressed as a BPMN model to merge it at runtime. There are not limits for the length or depth of model elements used in tactic models. For example, the collapsed Log Intruder’s Activities subprocess in Figure 11 contains the BPMN activities that describe the invocation of service operations for writing in the network and data logs.

Figure 11. Deception Tactic Model

The set of tactic models can be created with the STP BPMN Modeler or with the infrastructure provided by EMF. For instance, Figure 12 describes the EMF deception tactic model (expressed as a composition model in XMI format) in our case study. Since the STP metamodel dictates that model elements need to be contained in a pool, \textit{DeceptionTacticPool} was created to contain a subprocess with all the elements of this model.

Figure 12. Deception Tactic Model in XMI Format

5.4 Implement the Service Operations for Tactics

The objective of this task is to implement the service operations for each tactic (see Figure 13). The developer role is in charge of creating the atomic and composite Web services that implement the tactics.

Some tactics can be implemented as atomic Web services (their implementation is self contained and does not invoke any other services). Other tactics can be implemented as composite Web services. These tactics are more complex than the ones implemented as atomic Web services because they can invoke several service operations. Different tools (both free and commercial) can be used to facilitate the creation of Web services.

In the case of tactics that are based on composite Web services, Babel and our FMF-based configuration tool can be used to generate WS-BPEL code. First, Babel can partially generate the WS-BPEL code from tactic models expressed as BPMN composition models. Afterwards, our FMF-based configuration tool can be used to complete the WS-BPEL code.
5.5 Create the Fragments of WS-BPEL Code to Invoke Tactics

The objective of this task is to create the fragments of WS-BPEL code that will invoke the service operations implementing the tactics (see Figure 14).

The developer role is in charge of creating the fragments of WS-BPEL code to invoke tactics. Each tactic model maps to a WS-BPEL code fragment. Each WS-BPEL code fragment has an <invoke> instruction, which invokes the service operation implementing the tactic. At runtime, invocation instructions are put into the WS-BPEL composition schema in order to evolve the service composition.

A text editor can be used to implement the fragments of WS-BPEL code that invoke service operations implementing tactics. We propose to store each invocation instruction in a separated text file. Nevertheless, other approaches can also be followed to implement fragments of WS-BPEL code. For example, it is possible to store all invocation instructions in a database or to count on a single text file with all the invocation instructions. The key aspect in this task is to count on a fast mechanism to retrieve these instructions at runtime.

For instance, the following piece of WS-BPEL code can be used to invoke the deception tactic:

```
<bpel:invoke name="DeceptionTactic" partnerLink="DeceptionTacticPL" operation="deceit"
portType="ns17:deceptiontactic"
inputVariable="DeceptionTacticRequest"
outputVariable="DeceptionTacticResponse"></bpel:invoke>
```

5.6 Create the Rule Premises

The objective of this task is to create a rules file with rule premises. These rules are used at runtime to discover the requirements that can be affected by unknown context events (see Figure 15).

The systems analyst role is in charge of creating the rule premises and storing them in a rules file. The rules file works as a knowledge base to keep Resource Description Framework Schema (RDFS) rule premises. For example, \[R1: (\exists f \text{pre:serviceOperation} a) (\exists u \text{pre:rapidIncreaseResponseTimeInThreeMin} f) \implies (\exists u \text{pre:underAttack} a)\]. This rule means that if any Web service operation has rapidly increased its response time during the last three minutes, then it may be under attack.

At runtime, our approach evaluates arising context facts (i.e., context events) against these general rule premises in order to find the requirements that can be affected by unknown context events. A text editor can be used to implement the rules file.

For simple service compositions, rules can be obtained from human experts by: 1) collecting empirical data from the current service composition; 2) analyzing collected data to discover the symptoms of problematic situations; and 3) defining general situations in the context that can affect requirements. In complex service compositions, these steps can be extended with methods for generating rules from data (e.g. with heuristics or neural networks).

6. MECHANISMS FOR DYNAMIC EVOLUTION

The models created at design time are used to manage external uncertainty when facing unknown context events. The Evolution Planner and the Reconfiguration Engine are in charge of guiding the dynamic evolution of service
compositions. They are described in the following subsections.

6.1 Evolution Planner

The main objective of the Evolution Planner is to look for surviving tactics to protect the requirements that can be affected by problematic unknown context events. Therefore, the first two steps to trigger a dynamic evolution are as follows:

1. **To Observe the Context:** In order to collect context information, a context monitor (Alférez & Pelechano, 2011) periodically observes the context. It leverages the OWL Web Ontology Language to periodically insert new facts in an ontology that abstracts the context. **Individuals** in this ontology represent service operations. Each individual has **datatype properties** that are used to represent the current context state (e.g. the *isAvailable* datatype property indicates if a service operation is currently available).

2. **To Look for Unknown Context Events from the Collected Information:** In order to look for unknown context events, the Evolution Planner periodically checks the updated ontology. An observed context event is considered as **unknown** when there are not predefined context conditions to deal with it. **Context conditions** are Boolean expressions that work as SLAs (Alférez & Pelechano, 2011). If a context condition is fulfilled (i.e., an SLA is violated), then an adaptation is triggered on the service composition to deal with the arising situation (e.g. *UPSShipping, HasResponseTime, > 2,000 ms*). Our approach only observes changes in datatype properties of the ontology. In other words, we do not currently consider discovered service operations in the open world that may affect the structure of the ontology.

In order to face unknown context events, the Evolution Planner carries out the following steps:

1. **Search Affected Requirement(s):** In order to find the requirement(s) that can be affected by unknown context events, the Evolution Planner uses **forward chaining**. This method evaluates arising context facts (i.e., context events) against general rule premises in a knowledge base. A key advantage of forward chaining in the open world is that new context events can trigger new inferences.

   **Figure 16** shows a basic example when the unknown context event *F1* (a fact) is detected. In this case, rule *R1* has a condition that matches this new fact (step 1). Then, the forward chaining method fires the new fact *F2* (step 2). The process continues until the fact *F3* is fired (step 4). *F3* indicates that the *Barnes & Noble Books service operation can affect the High Security softgoal*. This example shows that evolutions are only triggered when requirements are negatively impacted.

2. **Search Surviving Tactics:** The objective of this step is to discover a tactic to preserve a requirement that can be negatively impacted by an unknown context event. To this end, the Evolution Planner carries out the following steps: 1) it looks for the affected softgoal in the requirements model; 2) it checks the claims associated to the affected softgoal and the set of tactics that depend on this softgoal; and 3) according to claims, it chooses the tactic with the most positive impact on the softgoal. For example, when the Evolution Planner finds that the *Barnes & Noble Books service operation can affect the High Security softgoal*, it looks for the High Security softgoal in the requirements model. According to claim C1 in **Figure 9**, the Evolution Planner chooses the Deception tactic, with the most positive impact on this softgoal.

![Figure 16. Forward Chaining Inference Example](http://hipore.com/ijsc)
6.2 Reconfiguration Engine

The main objective of the Reconfiguration Engine is to evolve the composition model with the tactic that has been found by the Evolution Planner. The Reconfiguration Engine carries out the following steps during evolution:

1. **Merge a Tactic Model into the Composition Model:**
   In order to inject the functionality of the discovered tactic into the service composition, it is necessary to: 1) identify a tactic model that describes the tactic to be triggered for preserving an affected requirement; and 2) to merge the required tactic model into the composition model to count on an enriched composition model that guides changes in the service composition. The merging operation was inspired by the *insert process fragment* pattern described in (Weber et al., 2008).

   The set of steps that are carried out in the merging operation are as follows (see Figure 17): 1) the activity (e.g. subprocess or task) describing the service operation that can negatively affect a requirement is put into a new subprocess; 2) the discovered tactic is put into the created subprocess; 3) a parallel relationship is created between the problematic activity and the tactic. As a result, the tactic's functionality will be executed when the problematic service operation is invoked; 4) the sequence flows that come in and go out from the problematic activity are redirected to the created subprocess. If subsequently, an activity that has been preserved with a tactic needs to be removed (i.e., in case of a triggered predefined dynamic adaptation), then the tactic is also removed. Figure 17 shows the evolved composition model after discovering the Barnes & Noble Books service operation can affect the High Security softgoal.

2. **Evolve the WS-BPEL Composition Schema:** The evolution of the WS-BPEL composition schema is guided by the information contained in the evolved composition model.

   We propose the following steps to reflect the changes in the evolved composition model into the WS-BPEL composition schema (see Figure 18): 1) the Reconfiguration Engine looks for the tactic that has been added into the composition model; 2) with this information, the Reconfiguration Engine looks for the WS-BPEL code fragment that invokes the tactical functionality. Each tactic model maps to a WS-BPEL code fragment, which is stored in a repository (i.e., a directory). Each code fragment has an associated WSDL, which is used to invoke the tactic's Web service; and 3) the Reconfiguration Engine injects the WS-BPEL code fragment that invokes the tactic into the composition schema. A parallel flow is dynamically

---

Figure 17. Evolved Composition Model in Our Running Example
created between the code that invokes the affected service operation and the code that invokes the tactic's Web service.

In each evolution, the Reconfiguration Engine puts the evolved composition schema and other required artifacts (e.g. WSDL files) into a deployment directory. This directory is hot deployed by the Execution Engine. Each new directory has a higher version to prevent the Execution Engine from deleting all the running instances with new deployments. New instances run according to the evolved composition schema. Therefore, our approach also covers the dimension of dynamic evolution of the composition schema. Existing approaches, such as (Weber et al., 2008), offer a solution to migrate running instances to cope with the evolved composition schema. Finally, instead of extending the functionality of the Execution Engine, our approach offers a transparent solution: it can be plugged/unplugged from the Execution Engine without modifying it.

7. Prototype and Evaluation

In our prototype, the models are specified in the XMI format to be queried at runtime. At runtime, SALMon (Ameller & Franch, 2008) inserts new facts into the ontology that represents the context. The Evolution Planner uses Jena 2\(^\text{12}\) for forward chaining. The Evolution Planner and the Reconfiguration Engine use the EMF Model Query 13 to carry out operations on models. The Reconfiguration Engine is implemented with our Model-based Reconfiguration Engine for Web Services (MoRE-WS) (Alférez & Pelechano, 2011). The Execution Engine was implemented with Apache ODE 14. Apache ODE was chosen because it offers mature hot-deployment support. Our prototype provides a graphical IDE to facilitate the creation and visualization of models (see Figure 19). A demonstration of our prototype and the models that were used in the running example can be found on our website 15.

We carried out the following set of experiments to evaluate the feasibility of our approach. These experiments were performed on a PC with an Intel Core 2 Duo 2.0 GHz processor and 4 GB RAM with Ubuntu version 10.04 and

---

\(12\) Jena: http://incubator.apache.org/jena/

\(13\) EMF Model Query: http://www.eclipse.org/modeling/emf/

\(14\) Apache ODE: http://ode.apache.org/

\(15\) Prototype: http://www.harveyalferez.com/dynamicevolutionservcomp/
7.1 Searching Affected Requirements

We evaluated the accuracy and performance of the inferences that look for the requirements that can be affected by unknown events. To this end, we purposely injected a set of events that were not predefined at design time to simulate uncertainty in the open world. We simulated some security attacks (e.g., DoS attacks), performance decrease in some service operations (by manually modifying the response times in the execution log), and the unavailability of other operations (by stopping some services). Figure 20 summarizes the results of 16 runs with increasing rules in the knowledge base and unknown events. Our approach found the affected requirements in 83.9% cases. The number of discovered affected requirements is directly proportional to the number of rules. This operation took 57 milliseconds in average.

7.2 Searching Surviving Tactics

We measured the response time when looking for surviving tactics in the requirements model. Since the response time to find a tactic in our running example took seven milliseconds on average, we decided to scale up the number of tactics in the requirements model. For 183 tactics, the response time was 63 milliseconds. In general, the response time of this operation is linear as the number of elements in the requirements model increases (see Figure 21).

7.3 Model-Based Evolution

We measured the response time of three key operations that are carried out on models during dynamic evolutions. In order to demonstrate that our approach scales well for large models, we randomly generated large composition models (see Figure 22).

First, the searching problematic service operation exhaustively navigates the composition model to look for an activity that describes a problematic service operation. Second, the adding tactic into the composition model operation looks for the activity that represents the affected service operation, adds the necessary model elements, and updates the composition model. Therefore, it took longer than the first operation. Third, the removing tactic from the composition model operation got a better response time because it only deletes the tactic-related elements and updates the model. Overall, even with a model population of 30,000 elements in the composition model, model operations had a good time response (< 300 milliseconds). It can be considered fast in the domain that we are addressing. Finally, our approach works well under stress circumstances.
Figure 21. Response Time when Searching Surviving Tactics

When several unknown events are achieved in tight time frames, the Reconfiguration Engine executes the required evolutions in the order in which events arrive.

7.4 Discussion

The evaluation demonstrated that model-driven dynamic evolution of service compositions has good performance and scales well. In order to increase the effectiveness of our approach in the open world, the number of rules in the knowledge base, and their related abstract requirements and surviving tactics have to be proportional to the complexity of the context of execution. Although our approach does not solve uncertainty completely, it is an important step towards uncertainty management.

8. RELATED WORK

Several research works related to autonomic service compositions have tended to implement variability constructs at the language level to guide dynamic adaptations in the closed world. For example, SCENE (Colombo et al., 2006) extends WS-BPEL with ECA rules that define consequences for conditions to guide the execution of binding and rebinding self-reconfiguration operations. VxBPEL (Koning et al., 2009) is an adaptation of WS-BPEL that allows variation points, variants, and configurations to be defined for a process in a service-centric system. In (Bairesi & Guinea, 2011), monitoring directives are expressed in the Web Service Constraint Language, and recovery strategies, which follow the ECA paradigm, are stated in the Web Service Recovery Language. Also, Aspect-Oriented Programming has been proposed to guide self-adaptive service compositions (Sonntag & Karastoyanova, 2011). However, implementing and managing dynamic adjustments at the language level can become complex as mentioned before. Our solution uses easy-to-understand models at runtime to guide the dynamic evolution of service compositions in the open world.

Traditional attempts to manage context-aware service compositions with models at runtime focus on dynamic adaptation in the closed world, not on dynamic evolution in the open world. For example, DySOA (Bosloper et al., 2005) offers components for monitoring and reconfiguring Web service-centric systems using models. QoSMOS (Calinescu, Grunse, Kwiakowskia, Mirandola, & Tamburrelli, 2011) combines formal specification of quality-of-service (QoS) requirements, model-based QoS evaluation, monitoring and parameter adaptation of the QoS models, and planning and execution of system adaptation. SASSY (Menasce et al., 2011) is a model-driven framework that provides runtime adaptation of service compositions in response to changing
There are several approaches that deal with modeling variability in service compositions that support BPs (Puhlmann, Schnieders, Weiland, & Weske, 2005; Rosemann & Van der Aalst, 2007; Gottschalk, van der Aalst, Jansen-Vullers, & Rosa, 2008). Although these works have inspired ours, they integrate all possible process variants in a single model. It results in large and difficult-to-understand models. On the contrary, we propose to reason about variable tactics separately and to leverage models at runtime to guide dynamic evolutions.

Recently, the community of models at runtime has shown interest on using models during execution to face uncertainty in the open world (Aßmann, Bencomo, Cheng, & France, 2011). However, just a small group of works, which are not focused on service compositions, deal with uncertainty by means of models at runtime (Welsh, Sawyer, & Bencomo, 2011; Cheng, Sawyer, Bencomo, & Whittle, 2009; Goldsby & Cheng, 2008). Moreover, the models that are proposed by these approaches do not evolve at runtime. Therefore, the capacity of reaction to manage unknown context events decreases because the initial models are unable to support them.

9. CONCLUSIONS AND FUTURE WORK

In this article, we have described a tool-supported approach that leverages models at runtime to guide the dynamic evolution of context-aware service compositions in the open world. Our approach can be used to manage uncertainty produced by unknown context events. The use of models at runtime has the following benefits: 1) modeling effort made at design time also provides a rich semantic base for autonomic behavior during execution; 2) since models are causally connected to the underlying service composition, they provide up-to-date information to drive subsequent evolutions; and 3) technological bridges are avoided because the model representations that are used at design time are kept at runtime. An evaluation demonstrated that model-driven dynamic evolution of service compositions has good performance and scales well.

As future work, we are going to use Constraint Programming to verify the evolved models and check that generated configurations respect the constraints imposed by the models. Also, we are going to verify that the introduction of tactics does not negatively affect other expected goals. In addition, we are going to propose a methodology to build and collect generic tactics.

10. ACKNOWLEDGEMENT

This work has been developed with the support of MINECO under the project SMART-ADAPT TIN2013-42981-P.

11. REFERENCES


http://hipore.com/ijsc


Authors

Germán Harvey Alférez

Salinas is a professor at the School of Engineering and Technology, Universidad de Montemorelos, Mexico. He holds a Ph.D. in Computer Science from Universitat Politècnica de València (Spain), a MSc in Information and Communication Technology from Assumption University (Thailand), and a BSc in Computer Science Engineering from Universidad EAFIT (Colombia). His research interests include Services Computing, Model-Driven Engineering, Models at Runtime, Autonomic Computing, and Dynamic Software Product Lines. He has contributed to publications in journals, book chapters, and international conferences, such as SPLC, ECOWS, MODELS, ICWS, and SERP. He has worked in universities, IT companies, and research groups on four continents (America, Asia, Australia, and Europe). www.harveyalferez.com.

Vicente Pelechano Ferragud is an associate professor at the Research Center on Software Production Methods (PROS) in the Universitat Politècnica de València (UPV) and leads the technical supervision of MOSKitt (http://www.moskitt.org), an open source CASE tool that gives full support to Model-Driven Software Development. His research interests include Model-Driven Engineering, Services Computing, Mobile and Ubiquitous Computing, Software Product Lines, Autonomous Computing, Human-Computer Interaction, and Business Process Modeling. Pelechano received his Ph.D. from the UPV in 2001. He has published articles in well-known scientific journals and international conferences. He has been member of scientific and organization committees of prestigious international conferences and workshops.

http://hipore.com/ijsc