

Technical Report 2015-001

# Integration of Reliability Prediction into Service Composition Frameworks

Thanh-Trung Pham and Alexander P. Vazhenin

September 14, 2015



School of Computer Science and Engineering  
The University of Aizu  
Tsuruga, Ikki-machi, Aizu-Wakamatsu City  
Fukushima 965-8580, Japan

<p>Title: Integration of Reliability Prediction into Service Composition Frameworks</p>	
<p>Authors: Thanh-Trung Pham and Alexander P. Vazhenin</p>	
<p>Key Words and Phrases: Reliability-aware service compositions, integration support of reliability prediction</p>	
<p>Abstract: Service composition frameworks hide the complexity of the underlying technologies and enable their users to easily compose new services from existing ones. To anticipate the reliability of composed services before their actual operations, it is possible for service composers to use existing model-based software reliability prediction techniques. However, such techniques either neglect or have only basic integration support into service composition frameworks. This limits their applicability because (1) neglecting integration support leads to high effort and specialist knowledge required to build models necessary for reliability predictions and (2) basic integration support likely reduces the flexibility and reusability when integrating into different service composition frameworks. In this paper, we present RMPI-SerComp to offer enhanced integration support into service composition frameworks for RMPI, a recent model-based software reliability prediction technique. Our approach enables service composition frameworks to achieve a systematic consideration of service reliability during composing, without the need for any specialist knowledge of the underlying prediction method. Via a case study, we validate the applicability of our approach by utilizing RMPI-SerComp to integrate the reliability prediction capability into a service composition framework.</p>	
<p>Report Date: September 14, 2015</p>	<p>Written Language: English</p>
<p>Any Other Identifying Information of this Report:</p>	
<p>Distribution Statement: First Issue: 3 copies</p>	
<p>Supplementary Notes:</p>	

Active Knowledge Engineering Laboratory  
 The University of Aizu  
 Aizu-Wakamatsu  
 Fukushima 965-8580  
 Japan

# Integration of Reliability Prediction into Service Composition Frameworks

Thanh-Trung Pham and Alexander P. Vazhenin

School of Computer Science and Engineering  
The University of Aizu (UoA)  
Aizu-Wakamatsu, Fukushima, Japan  
Email: {pttrung,vazhenin}@u-aizu.ac.jp

**Abstract**—Service composition frameworks hide the complexity of the underlying technologies and enable their users to easily compose new services from existing ones. To anticipate the reliability of composed services before their actual operations, it is possible for service composers to use existing model-based software reliability prediction techniques. However, such techniques either neglect or have only basic integration support into service composition frameworks. This limits their applicability because (1) neglecting integration support leads to high effort and specialist knowledge required to build models necessary for reliability predictions and (2) basic integration support likely reduces the flexibility and reusability when integrating into different service composition frameworks.

In this paper, we present RMPI-SerComp to offer enhanced integration support into service composition frameworks for RMPI, a recent model-based software reliability prediction technique. Our approach enables service composition frameworks to achieve a systematic consideration of service reliability during composing, without the need for any specialist knowledge of the underlying prediction method. Via a case study, we validate the applicability of our approach by utilizing RMPI-SerComp to integrate the reliability prediction capability into a service composition framework.

**Index Terms**—Reliability-aware service compositions, integration support of reliability prediction.

## I. INTRODUCTION

Software providers and consumers have gained more and more interest in service-oriented software systems. For both sides, this can help to reduce build-up time and infrastructure cost, as well as to advantage management [1]. Besides single services, service providers also offer composed services on demand, under customer requests. In order to enable intuitive creation of composed services without specialist knowledge of the underlying technologies, service composition frameworks (e.g. [2]–[4]) have come into view.

Besides functional properties (e.g. correctness), quality properties (e.g. reliability, performance, security, etc.) are becoming more important to attain and assure the acceptance of users. Analyzing reliability of composed services (i.e. the probability of failure-free operation in a given time span) is apparently a challenge because:

- Services can be composed in a dynamic manner, leading to multiple compositions to be analyzed.
- Creating and testing each possible composition requested by customers are usually not feasible as it is resource- and time-consuming to conduct on all compositions and/or invoking services may be charged.

Using existing model-based software reliability prediction techniques is a potential solution to this problem. Being based on the service model and its reliability-influencing factors, they use analytical calculations or simulations to predict the service reliability, without any actual executions of the service.

However, many existing model-based software reliability prediction techniques (e.g. [5]–[7]) are missing integration support into service composition frameworks. As a result, service composers have to apply these prediction techniques separately, via manually specifying the input information necessary for reliability predictions. This manual specification is laborious, error-prone and not always possible because of the missing knowledge regarding the inner characteristics of the service model. Moreover, for many prediction techniques, service composers are required to have specialist knowledge (e.g. Markov models) and to understand the details of the prediction techniques (e.g. the mappings of services or their internal behavioral states to Markov states). Manual reliability prediction is ineffective and difficult to accomplish for service composers, especially when it is to be done repeatedly for composed services requested by customers and their possible composition alternatives. On the other hand, existing techniques (e.g. [8]) that offer basic integration support typically lack a public Application Programming Interface (API) and a plug-in architecture for their integration support. These two factors are necessary to achieve the flexibility and reusability when integrating into different service composition frameworks. By utilizing the public API, integrations can be adapted according to integration requirements. The plug-in architecture enables sharing and reusing integration parts.

**Contribution:** In this paper, we present the RMPI-SerComp<sup>1</sup> to offer enhanced integration support into service composition frameworks for RMPI<sup>2</sup> [9], our recent model-based software reliability prediction technique. With RMPI-SerComp, the standard service composition process of service composition frameworks can be enhanced to become a reliability-aware service composition process, which includes reliability prediction for composed services as well as supports the decision between different composition alternatives. RMPI-SerComp offers a public API and a plug-in architecture for flexible and reusable integrations into service composition frameworks. RMPI-SerComp is realized via a set of components and their

<sup>1</sup>SerComp: Service Compositions

<sup>2</sup>RMPI: Reliability Modeling, Prediction, and Improvements

interactions. We validate our approach in a case study where we utilize RMPI-SerComp to enhance a service composition framework with the reliability prediction capability.

*Structure:* The rest of this paper is organized as follows: Section II introduces the existing foundations of our approach. Section III surveys related work. Section IV describes our recommended reliability-aware service composition process. Section V describes in detail RMPI-SerComp’s architecture and its workflow. Section VI demonstrates our approach with a case study. Section VII discusses our assumptions and limitations, and Section VIII concludes the paper.

## II. FOUNDATIONS

In this section, we briefly introduce the main related research areas of our approach: service compositions and model-based software reliability prediction. Service compositions provide a flexible way for software creation, but it is also challenging to assure the reliability of dynamically composed services. Model-based software reliability prediction offers the ability to model and predict the reliability of software systems, which can be embedded and utilized throughout a software engineering process.

### A. Service Compositions

Software service is a self-describing entity that encapsulates its contents (i.e. functionalities and data) and is accessible via its interface [1]. This information is sufficient for service compositions, i.e. orchestrating existing services to provide more functionalities as new services. A workflow<sup>3</sup> is often used to describe the composition of services, using programming or modeling languages such as WS-BPEL (Web Services Business Process Execution Language) [1]. In recent years, service composition frameworks (e.g. BizTalk Server [2], Oracle BPM [3], jBPM [4]) have been developing to make service compositions easier for users with little or even zero programming experience through intuitive and flexible development environments. In order to assess the value of composed services, it is necessary to evaluate not only their functional properties but also their quality properties [12].

### B. Model-based Software Reliability Prediction

Fig. 1 depicts a general model-based reliability prediction process [13]–[15]. It starts with the model of a software system. Usually, this model exists as a part of a software engineering process and typically, is expressed using design-oriented modeling languages, e.g. UML (Unified Modeling Language), ADL (Architecture Description Language).

Predicting reliability requires additional data which is not yet present in the system model. Therefore, in Step 1, the modeler needs to annotate the system model with reliability-relevant data. The amount of additional data is highly dependent on the concrete prediction approaches. In general, the annotations include failure possibilities in the system and usage characteristics of the system.

<sup>3</sup>Basically, any other kinds of glue code could also be used, e.g. [10], [11].

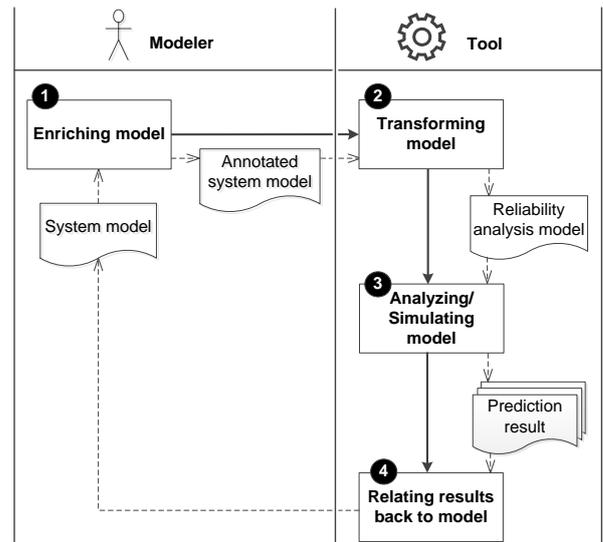


Fig. 1. Model-based software reliability prediction.

In Step 2, the annotated system model is transformed into a reliability analysis model, e.g. a Markov model, a Bayesian network. The analysis model is then solved by an analyzer or a simulator to obtain prediction results in Step 3. The set of available results is dependent on the modeling formalisms as well as its solver.

Finally, the prediction results are related back to the beginning system model in Step 4. This feedback is supposed to support the modeler in answering evaluation questions, e.g. which parts of the system are most likely to cause failures?. The last three steps should be performed automatically by tools.

## III. RELATED WORK

Seminal work in software reliability engineering [16] focuses on system testing and reliability growth models treating systems as black boxes. Recently, many model-based software reliability prediction approaches have been proposed [13]–[15], treating systems as compositions of components or services. In the following, we examine these approaches with respect to integration support into service composition frameworks, targeting at automated and transparent-to-service-composers reliability predictions in these frameworks.

Cheung’s approach [17], a well-known representative of the field, expresses the control flow between components in a software system using an absorbing Discrete-Time Markov Chain (DTMC) and encodes the usage profile of the system into their transition probabilities. The approaches of Cortellessa et al. [18] and Filieri et al. [19] build on the same formalism and superimpose error propagation models. The approach of Wang et al. [20] extends the formalism to capture heterogeneous software architectures incorporating different architectural styles. Further approaches building on the Cheung’s model include the approach of Sharma et al. [21] which takes into consideration the possibilities of component

restarts and application retries, the approach of Sharma et al. [22] which offers combined predictions of multiple quality attributes, the approach of Lipton et al. [23] which conducts reliability optimization. These approaches are tailored to component-based software systems and disregard integration support into service composition frameworks.

The approach of Reussner et al. [24] is based on Rich Architecture Definition Language (RADL) but employs the same underlying theory as Cheung’s approach for reliability predictions. The approach of Reussner et al. can be seen as a precursor of the approach of Brosch et al. [25] which offers a parameterized reliability prediction taking into consideration the influences of the system’s usage profile and the system’s execution environment. The approach of Cheung et al. [26] is specifically tailored to component-level reliability prediction. These approaches also disregard integration support into service composition frameworks.

Scenario-based approaches such as the approach of Yacoub et al. [27] which constructs component dependency graphs from component sequence diagrams as a basic for reliability predictions, the approaches of Cortellessa et al. [28], Popic et al. [29], and Goseva et al. [30] which employ UML diagrams annotated with reliability properties, the approach of Rodrigues et al. [31] which is based on message sequence charts, also disregard integration support into service composition frameworks.

The approaches of Grassi [5], Cortellessa et al. [6], and Zheng et al. [7] aim at reliability prediction for service-oriented architectures (SOA). The approaches of Grassi and Cortellessa et al. consider recursively composed services, where each service may invoke multiple external services in order to complete its own execution. The approach of Zheng et al. employs a workflow description for composite services with sequential, looping, and parallel structures. The approach of Grassi et al. [32] reuses a number of concepts of the approach of Grassi [5] and proposes the Kernal Language for Performance and Reliability Analysis (KLAPER). However, these approaches do not consider automated and transparent-to-service-composers predictions in service composition frameworks.

The approach of Klatt et al. [8] considers integration support into service composition framework and is currently based on the approach of Brosch et al. for reliability predictions. However, it does not consider a public API and a plug-in architecture for their integration support, which are the two important factors to offer the flexibility and reusability when integrating into different service composition framework.

We described RMPI, which is our recent model-based software reliability prediction technique and do not related to integration support into service composition frameworks, in our former work [9], [33].

#### IV. RELIABILITY-AWARE SERVICE COMPOSITION PROCESS

In this section, we present our recommended reliability-aware service composition process. It is enhanced from the

standard service composition process [1] to include general steps to create composed services that satisfy the given reliability requirements and to regard generic reliability-influencing factors of composed services.

At the generic level, the reliability of a composed service is dependent on: (1) its *composition workflow* (i.e. the control and data flow between involved services), (2) the failure possibilities of *required external services*, and (3) its *usage profiles* (e.g. how often the composed service is used).

At the essence level, similar to any other kinds of software systems, faults in the implementations of required external services are the root causes for failures of the composed service and error propagation in the composition workflow is crucial in the chain leading to failures of the composed service [34]. Besides basic control flow structures (i.e. sequential, branching, or looping structures), the composition workflow of a composed service can employ parallel structures and fault tolerance structures to improve the service performance and the service reliability, respectively. Employing parallel structures could lead to *concurrent error propagation* paths as well as *concurrently present errors* and employing fault tolerance structures could involve non-trivial *error detection*, *error handling*, and *fault handling* activities [9].

Notice that the underlying model-based software reliability method should be chosen in such a way that it explicitly take all the factors mentioned above into consideration for accurate prediction results.

Fig. 2 describes our recommended process with six steps, including familiar steps of the standard service composition process (shown in grays). The service composer (i.e. the user of a service composition framework) is responsible to conduct these steps. The top of the figure shows the customer’s requirements that the composed service must satisfy under the given usage profiles. Artifacts related to external services (e.g. their descriptions and failure models expressing failure possibilities) are shown at the bottom of the figure.

Step 1 is to create an abstract service composition workflow starting from the scratch or selecting from a repository of existing workflows. To realize the composed service, the abstract composition workflow includes abstract dependencies on external services and possibly sets of candidates that match these dependencies. There might be multiple candidates available for each dependency but with different failure models. Fig. 3a shows an example of a simple abstract composition workflow, including a set of abstract service dependencies ( $asd_1, \dots, asd_4$ ), their candidates, and basic control flow structures (a sequential structure, a branching structure with branching conditions, and a looping structure with a loop count).

From the abstract workflow, the service composer selects a candidate for each abstract dependency, yielding the concrete service composition workflow in Step 2. Fig. 3b shows an example of a simple concrete composition workflow with selected external services ( $s_{11}, \dots, s_{42}$ ).

In Step 3, from all the relevant information in the usage profiles, the concrete service composition workflow, the failure

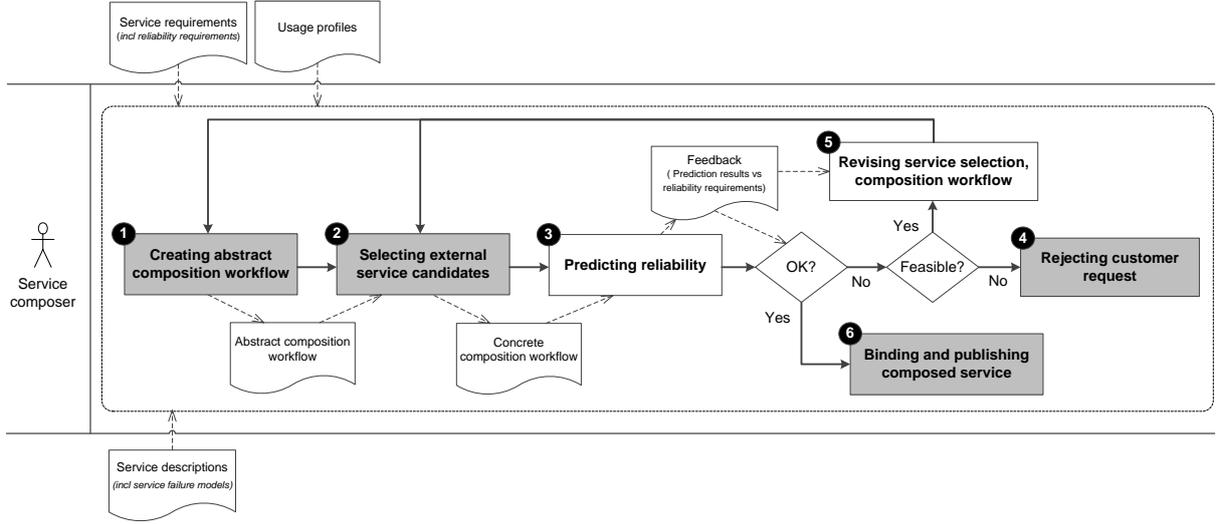


Fig. 2. Reliability-aware service composition process.

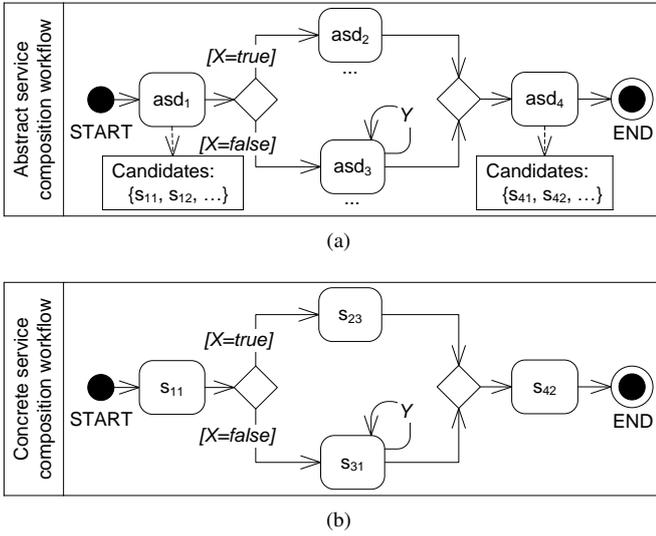


Fig. 3. Service composition workflows: (a) Abstract and (b) Concrete.

models of external services, and the reliability requirements, it is possible to automatically derive a reliability prediction model which, in turn, is used by an automated reliability prediction to provide feedback regarding the possibility of the service composition to satisfy the customer’s reliability requirements. Because of these automations, the service composer does not need to have any specialist knowledge of the underlying reliability prediction method.

If the feedback shows that the customer’s reliability requirements can be met, Step 6, binding and publishing the composed service, is performed. Otherwise, the service composer can revise the service selection (e.g. selecting other external service candidates with different failure models), or the composition workflow (e.g. introducing fault tolerance structures) in Step 5 and then repeat the prediction. The feedback can also include information to guide the revising process, e.g.

identifying the most critical parts of the composition workflow. All of these may be repeated multiple times, without any actual executions of the composed service. If the composer cannot create any feasible offers in reply to the customer’s request, he rejects the request as in Step 4.

## V. RMPI-SERCOMP

In this section, we describe RMPI-SerComp. It offers enhanced integration support into service composition frameworks for RMPI [9]. With RMPI-SerComp, service composition frameworks can achieve a systematic consideration of service reliability during composing, according to our recommended reliability-aware service composition process.

Initially, RMPI was tailored towards component-based software systems. However, it can also be used to provide reliability modeling and prediction for composed services, explicitly taking into consideration all the reliability-influencing factors mentioned in Section IV. It has been realized in RMPISchema and RMPITool. RMPISchema is a developer-friendly reliability modeling language to capture comprehensively different reliability-influencing factors into a reliability model of the system under study. RMPITool offers an automated evaluation of the system reliability model to obtain prediction results. RMPI has been successfully evaluated in several case studies [9], [33].

### A. Architecture and Prediction Workflow

Fig. 4 shows the architecture of RMPI-SerComp, including existing components of RMPI (shown in gray) and newly added components. For standalone reliability predictions, the software architect uses RMPISchema to create a reliability model of the system under study. Based on the created system reliability model, he manually interacts with RMPITool via a command line interface (CLI) (provided by component RMPICLI) to validate the model and then to obtain prediction results by activating analytical calculations (provided

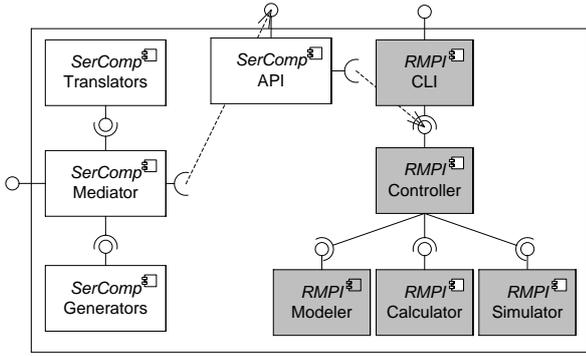


Fig. 4. RMPI-SerComp's architecture.

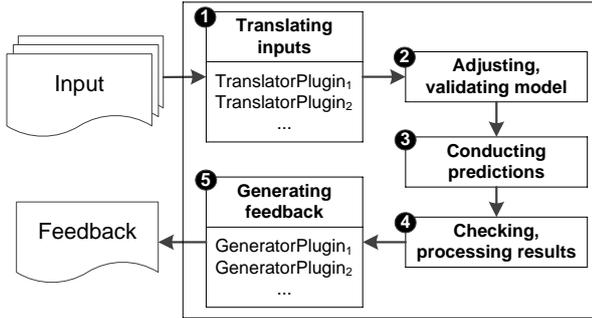


Fig. 5. RMPI-SerComp's prediction workflow.

by component *RMPICalculator*) or simulations (provided by component *RMPISimulator*). As an alternative to the manual interaction, component *SerCompAPI* has been developed to offer a public API covering all the capabilities of RMPI (i.e. generating and validating system reliability models, conducting predictions, and obtaining prediction results).

As a user of service composition frameworks, the service composer expects the transparency of reliability prediction details. From his viewpoint, the inputs necessary for the prediction are specified in a user-friendly way via the user interfaces of the service composition frameworks. Internally, these inputs are translated into a valid input format for the underlying reliability prediction method. In case of RMPI-SerComp, in order to enable multiple framework integrations, different input translators can be developed by utilizing the API provided by component *SerCompAPI* and then registered as plug-ins in component *SerCompTranslators* via component *SerCompMediator*.

The service composer also expects that the prediction results are returned as feedback in a understandable and expressive form, e.g. saying that the given reliability requirements cannot be met, representing prediction results as a graph. Therefore, similar to the case of input translators, it is possible to develop multiple feedback generators for different presentations of prediction results as feedback for the service composer and register them as plug-ins in component *SerCompGenerators*.

Fig. 5 shows the prediction workflow of RMPI-SerComp with five internal steps. These steps are controlled and carried

out automatically by component *SerCompMediator*. First, the inputs provided by the composition framework and its user are translated by the chosen translator plug-in into an input model for RMPI. As a translator plug-in can produce an unexpected input model, it is necessary for this input model to be adjusted (optionally) and validated to make sure that it is in a valid form with all the relevant information.

Based on the valid input model, prediction results are obtained by conducting predictions via analytical calculations or simulations. Because it is possible to use any of the two methods, as well as to use simulations multiple times, the next step is to check and process these results (e.g. computing mean values) before delivering them to generator plug-ins to create presentations of prediction results as feedback for the service composer.

**Remark:** With its public API and plug-in architecture, RMPI-SerComp allows developing and employing different input translators and feedback generators, offering flexible and reusable integrations into service composition frameworks for RMPI. This plug-in architecture also decouples service composition frameworks from RMPI, supporting individual changes from both sides. For example, if the format of the inputs from a service composition framework changes, only the corresponding input translator needs to be updated while RMPI stays unchanged. As another example, if prediction results from RMPI change their formats, it is required to update only feedback generators.

## B. TranslatorPlug-ins

An input translator plug-in is to translate inputs from a service composition framework into a valid input model for RMPI (i.e. input model of an equivalent component-based software system). These inputs include: (1) reliability requirements for the composed service, (2) usage profiles for the composed service, (3) the concrete service composition workflow of the composed service, (4) failure models of external services required by the composition workflow, and (5) run configuration settings, e.g. method for reliability predictions (analytical calculation or simulation), maximum simulation time. Formats of these inputs depend on concrete service composition frameworks and can be proprietary or standard formats (e.g. XML - eXtensible Markup Language, BPMN - Business Process Model and Notation). It is the task of framework integrators to develop input translator plug-ins.

Let  $\mathcal{SR}$  be the set of reliability requirements for the composed service,  $\mathcal{UP} = \{up_i\}$  with  $i \in \{1, 2, \dots, m\}$  be the set of  $m$  usage profiles for the composed service,  $sn$  be the composed service's name,  $CW$  be the concrete service composition workflow of the composed service,  $\mathcal{XS} = \{s_j\}$  with  $j \in \{1, 2, \dots, n\}$  be the set of  $n$  required external services for the composed service, and  $\mathcal{FM} = \{fm_j\}$  with  $j \in \{1, 2, \dots, n\}$  be the failure models of the required external services where  $fm_j$  is the failure model of required external service  $s_j$ ,  $\mathcal{RCS}$  be the set of run configuration settings.

Then, an input translator plug-in translates these inputs into an input model of an equivalent component-based software

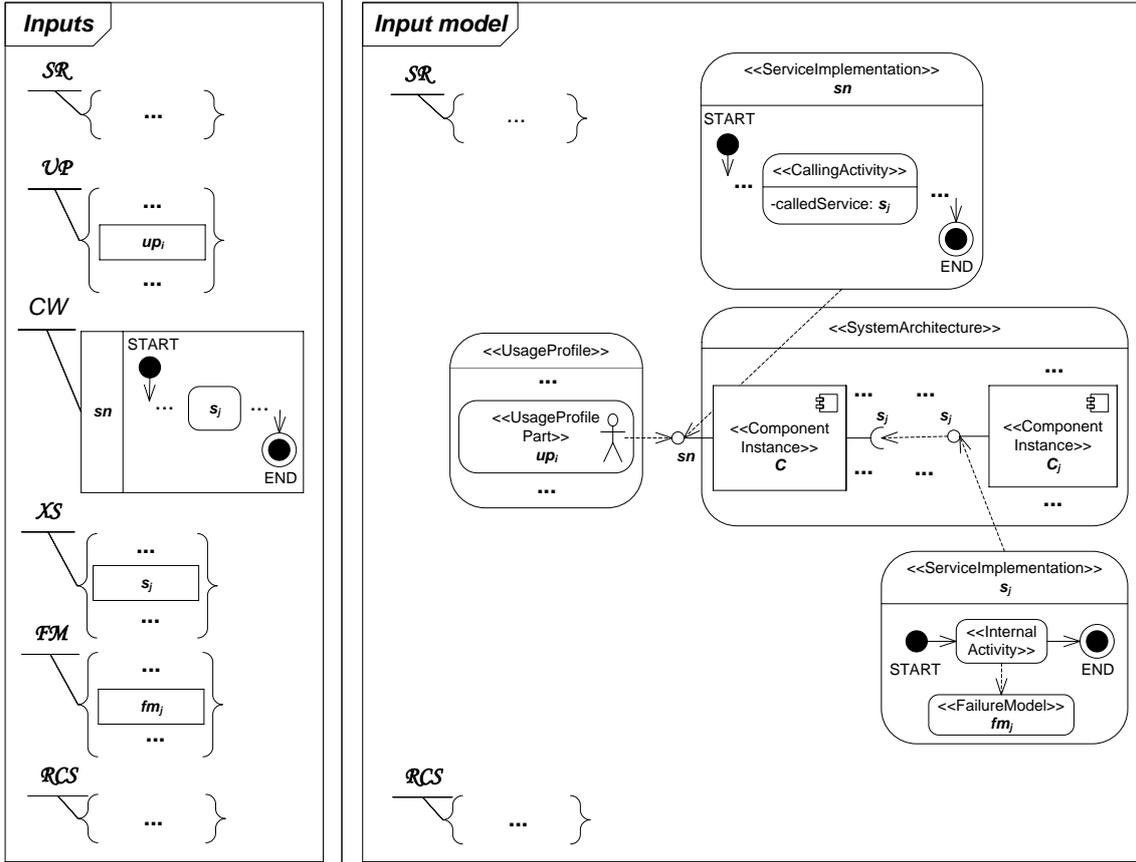


Fig. 6. A translator plug-in's translation.

system for RMPI by utilizing the public API of RMPI-SerComp as follows (see also Fig. 6):

- For each required external service  $s_j \in \mathcal{XS}$ , one component  $C_j$  is created in the resulting input model. Component  $C_j$  provides one service  $s_j$ . The implementation of provided service  $s_j$  includes one internal activity. The activity has  $fm_j \in \mathcal{FM}$  as its failure model.
- One component  $C$  is created in the resulting input model. Component  $C$  requires  $n$  services  $s_j$  with all  $j \in \{1, 2, \dots, n\}$  and provides one service  $sn$ . The implementation of provided service  $sn$  has its flow similar to the flow of composition workflow  $CW$ . For each dependency on external service  $s_j$  in composition workflow  $CW$ , there is a calling activity at the corresponding position in the implementation of provided service  $sn$  to call required service  $s_j$  of component  $C$ .
- A system architecture is created in the resulting input model. Within it, there are one component instance of component  $C$  and  $n$  component instances of  $n$  component  $C_j$  for all  $j \in \{1, 2, \dots, n\}$ . There are also component connectors from provided service  $s_j$  of the instance of component  $C_j$  to required service  $s_j$  of the instance of component  $C$  for all  $j \in \{1, 2, \dots, n\}$ . Provided service  $sn$  of the instance of component  $C$  is exposed as a user interface for users.

- A usage profile is created in the resulting input model. It has  $up_i \in \mathcal{UP}$  as its usage profile part for all  $i \in \{1, 2, \dots, m\}$ . All of the usage profile parts refer to the user interface.
- The set of reliability requirements for the composed service,  $SR$ , and the set of run configuration settings,  $RCS$ , are simply passed through.

**Remark:** The expressiveness of the public API of RMPI-SerComp is equal to that of RMPISchema (i.e. the reliability modeling language of RMPI) with regard to control flow structures, failure models, and usage profiles. Therefore, it is possible to support the translation for multiple control flow structures (including sequential, branching, looping, parallel, and different fault tolerance structures) in service composition workflows, comprehensive failure models (with multiple failure types, concurrent present errors, error propagation, etc.) of external services, and flexible usage profiles of composed services. The case study in Section VI shows several examples of inputs from a service composition frameworks.

### C. GeneratorPlug-ins

A feedback generator plug-in is to generate a presentation of prediction results as feedback for the service composer. Framework integrators are responsible to develop feedback generator plug-ins. Prediction results delivered to a generator

Reliability requirements	Satisfy?	Prediction results
Reliability > 0.993	(o)	0.994 = Reliability
Failure probability of $\{F_1\}$ < 0.003	(o)	0.002 = Failure probability of $\{F_1\}$
Failure probability of $\{F_2\}$ $\leq$ 0.002	(o)	0.001 = Failure probability of $\{F_2\}$
		0.003 = Failure probability of $\{F_1, F_2\}$
<b>All reliability requirements are satisfied.</b>	(o)	

Fig. 7. A tabular presentation of prediction results.

plug-in by RMPI-SerComp include not only the predicted reliability of the composed service but also its predicted failure probabilities of different failure types.

Let  $\mathcal{AS}$  be the set of all possible sets of failure types of the composed service,  $fp(F : F \in \mathcal{AS})$  be the predicted failure probability of failure types  $F$  of the composed service,  $R$  be the predicted reliability of the composed service,  $\mathcal{PR}$  be the set of prediction results for the composed service, then  $\mathcal{PR} = \{R, fp(F) : \forall F \in \mathcal{AS}\}$  such that  $R + \sum_{\forall F \in \mathcal{AS}} fp(F) = 1$ .

Considering reliability requirement  $sr \in \mathcal{SR}$  for the composed service, then  $sr$  belongs to either *Type I* or *Type II*:

- *Type I* - A requirement for the reliability of the composed service: *Reliability* > /  $\geq c_R$  with  $0 \leq c_R \leq 1$  (i.e. the reliability of the composed service is required to be greater than/greater than or equal to  $c_R$ , respectively).
- *Type II* - A requirement for the failure probability of failure types  $F$  of the composed service: *Failure probability of  $F$*  < /  $\leq c_F$  with  $0 \leq c_F \leq 1$  (i.e. the failure probability of failure types  $F$  of the composed service is required to be less than/less than or equal to  $c_F$ , respectively).

Then, by comparisons between reliability requirements  $sr \in \mathcal{SR}$  with the corresponding prediction results  $pr \in \mathcal{PR}$ , it is possible for a feedback generator plug-in to derive different presentations of prediction results (e.g. textual, tabular, or graphical presentation):

- In case  $sr$  belongs to *Type I*: if  $R > / \geq c_R$  (i.e. the predicted reliability of the composed service is greater than/greater than or equal to  $c_R$ , respectively),  $sr$  is satisfied. Otherwise,  $sr$  cannot be met.
- In case  $sr$  belongs to *Type II*: if  $fp(F) < / \leq c_F$  (i.e. the predicted failure probability of failure types  $F$  of the composed service is less than/less than or equal to  $c_F$ , respectively),  $sr$  is satisfied. Otherwise,  $sr$  cannot be met.
- The set of reliability requirements  $\mathcal{SR}$  is satisfied if all  $sr \in \mathcal{SR}$  are satisfied. Otherwise,  $\mathcal{SR}$  cannot be met.

Fig. 7 shows an example of a tabular presentation of prediction results. In this example,  $\mathcal{AS} = \{\{F_1\}, \{F_2\}, \{F_1, F_2\}\}$  (where  $\{F_1, F_2\}$  is the concurrent presence of two failure types

$F_1$  and  $F_2$ ),  $\mathcal{SR}$  including three reliability requirements is shown in column 1, and  $\mathcal{PR}$  including four prediction results is shown in column 3. There is no requirement related to the failure probability of failure types  $\{F_1, F_2\}$  (a blank cell at column 1 row 5). Column 2 shows satisfaction states of reliability requirements. Because all requirements are satisfied,  $\mathcal{SR}$  is satisfied as summarized in the last row. The case study in Section VI also shows different possible presentations of prediction results.

**Remark:** Besides the standard integration scenario described above, based on the inputs from service composition frameworks, multiple input models for RMPI can be created by an input translator plug-in to reflect a family of similar service compositions or to vary values of the parameters in the inputs, accounting for existing uncertainties of the parameters. As a result, multiple corresponding sets of prediction results are delivered to feedback generator plug-ins, which can be based on to generate presentations for supporting the decision between the similar service compositions or studying the effects of the variations of the parameters in the inputs to the prediction results (a.k.a sensitivity analyses).

#### D. Implementation

We have implemented RMPI-SerComp as a Java library. As described above, framework integrators can utilize this library to develop different translator and generator plug-ins as well as to programmatically trigger RMPI-SerComp's prediction workflow. The library also supports publishing the prediction workflow as a SOAP-based web service. This service can be invoked remotely and returns presentations of prediction results to its callers.

The implementations of RMPI-SerComp is open source and available at our project website [35].

## VI. CASE STUDY EVALUATION

### A. Evaluation Goals

This section serves to validate the applicability of our approach. The goals of the validation are to (i) demonstrate RMPI-SerComp's capability in integrating the reliability prediction according our recommended process (see Section IV) into a service composition framework and (ii) provide a rationale for the validity of RMPI-SerComp's prediction workflow.

### B. Framework Integration

The system chosen for the case study is a workflow system. It allows designing and executing workflows in which different activities (e.g. web services, Java objects, published workflows, ...) are orchestrated to solve business problems.

Although different activity types can be employed in a workflow, in this case study, we focus on the parts of the system (i.e. its architecture and functionality) that are involved in compositions of web services and therefore describe only these parts. Fig. 8 shows an extract of the involved system architecture. Component *ActivityRepository* manages information about the existing web services including external web services as well as internal ones (i.e. workflows published as

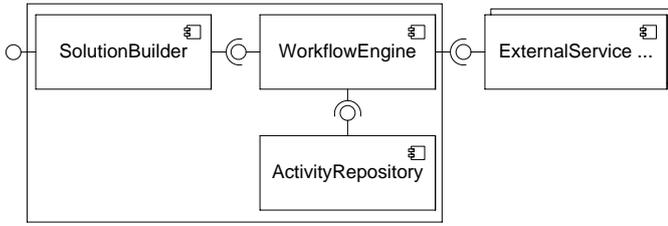


Fig. 8. The workflow system's architecture

web services). In addition to the workflow management capability, component *WorkflowEngine* also provides an execution environment for executing, testing, and debugging workflows. Component *SolutionBuilder* enables composing from existing web services via its user interface.

The workflow system does not include any support for reliability predictions. We utilized the capability of the system itself to develop two supporting workflows:

- Workflow *inputFailureModels* is used to provide additionally failure models for existing web services.
- Workflow *inputReliabilityRequirementsAndUsageProfiles* is used to provide additionally reliability requirements and usage profiles for the composed service.

Utilizing RMPI-SerComp as the Java library<sup>4</sup>, we developed and then registered with the library four prototype plug-ins: one input translator plug-in, named *WorkflowTranslator*, and three feedback generator plug-ins, named *TextualGenerator*, *TabularGenerator*, and *HybridGenerator*. The translator plug-in is to translate XML-based inputs (see Section V-B) from the workflow system into a valid input model for RMPI. The three generator plug-ins are to generate textual, tabular, and hybrid presentations of the prediction results, respectively (see also Section VI-C).

We also utilized the library to publish RMPI-SerComp's prediction workflow as a SOAP-based web service and developed one more supporting workflow, named *predictReliability*, for invoking this web service to obtain different presentations of prediction results. With all the three supporting workflows, our recommended reliability-aware service composition process (see Section IV) has been realized in the workflow system as shown in Fig. 9 (Several parts of the process are omitted for the sake of clarity). In the process, the service composer uses workflow *predictReliability* to obtain a textual, tabular, or hybrid presentation of prediction results of the composed service as feedback. In other words, the workflow system has been integrated with the reliability prediction capability. As a result, RMPI-SerComp has demonstrated its capability in integrating the reliability prediction according our recommended process into a service composition framework (Goal (i) in Section VI-A).

### C. Validity of RMPI-SerComp's Prediction Workflow

To achieve Goal (ii) in Section VI-A, we take the reporting service of a document exchange server, which allows

<sup>4</sup>For a full documentation and examples, refer to our project website [35]

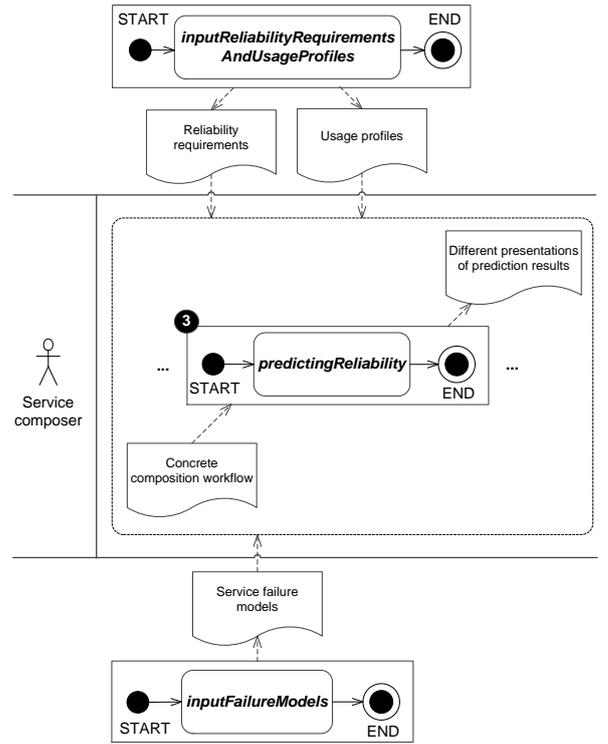


Fig. 9. Reliability-aware service composition process in the workflow system.

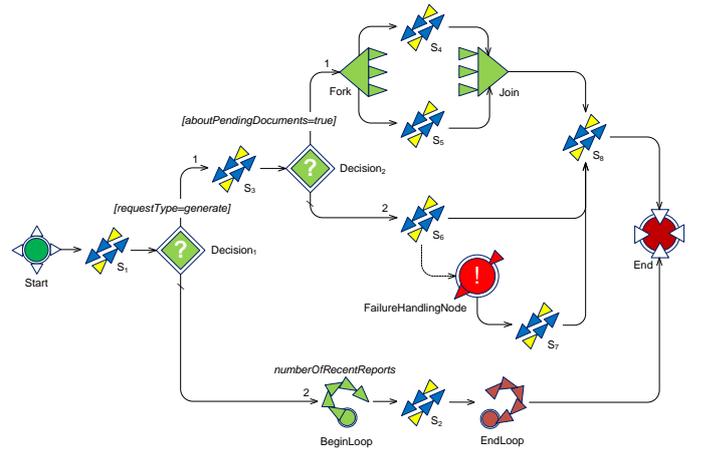


Fig. 10. The reporting service in the workflow system's modeling notation.

generating reports about pending or released documents. The reliability of this service and its variants has been analyzed in our former study [9] using RMPI.

Using the modeling notation of the workflow system (i.e. using service composition workflows), Fig. 10 presents the reporting service's variant in which only the multi-try-catch fault tolerance mechanism is used (i.e. variant *Only MTCS* in the former study) as an example. Services from  $S_1$  to  $S_8$  in this service composition workflow correspond to internal activities from  $a_1$  to  $a_8$  in component implementations of the reporting service when modeled using RMPI.

Service  $S_1$  handles incoming report requests from clients.

TABLE I  
TWO USAGE PROFILES FOR THE REPORTING SERVICE.

No.	Probability	Information
1	0.8	$p(\text{requestType}=\text{generate})=0.78$ $p(\text{aboutPendingDocuments}=\text{true})=0.56$ $\text{average}(\text{numberOfRecentReports})=2$
2	0.2	$p(\text{requestType}=\text{generate})=0.25$ $p(\text{aboutPendingDocuments}=\text{true})=0.47$ $\text{average}(\text{numberOfRecentReports})=4$

TABLE II  
RELIABILITY REQUIREMENTS FOR THE REPORTING SERVICE.

Reliability requirements
Reliability $\geq 0.996$
Failure probability of $\{FPI, FP2\} < 0.0006$
Failure probability of $\{FS2\} \leq 0.0006$

Gateway  $Decision_1$  operates as a branching structure: if an incoming request is about generating a new report ( $\text{requestType}=\text{generate}$ ), the gateway directs the control and data flow along its above link; otherwise (i.e. the incoming request is about viewing recently generated reports), the gateway directs the flow along its below link. Two gateways  $BeginLoop$  and  $EndLoop$  form a looping structure with  $numberOfRecentReports$  as its loop count and using service  $S_2$  to get a recently generated report. Service  $S_3$  generates the skeleton for the requested report. Gateway  $Decision_2$  directs the control and data flow along its above link if the requested report is about pending documents ( $\text{aboutPendingDocument}=\text{true}$ ); otherwise the gateway directs the flow along its below link. Gateways  $Fork$  and  $Join$  form a parallel structure using services  $S_4$  and  $S_5$  to get information about pending documents which are attached in emails and stored in file systems, respectively. Service  $S_6$  gets information about released documents from the logs. Gateway  $FailureHandlingNode$  has the ability to handle a signaled failure of service  $S_6$  and then directs the control and data flow to service  $S_7$  to get information about released documents from the database instead. With all the necessary information about documents, service  $S_8$  fills the content for the requested report.

For this case study, we reused the failure models of corresponding internal activities from the former study as the failure models for services from  $S_1$  to  $S_8$ . We also created two usage profiles to model the same usage scenarios of the reporting service as in the former study. Table I shows these usage profiles. They mean that with probability 0.8, clients access the reporting service with usage profile 1 and with probability 0.2, they access with usage profile 2. Each usage profile contains two probabilities and one average to determine the branching probabilities at two gateways  $Decision_1$  and  $Decision_2$ , and the average number of loops at gateway  $BeginLoop$ .

We set the reliability requirements for reporting service as in Table II. The reliability of the reporting service is required to be greater than or equal to 0.996. It is required that failure probability of failure types  $\{FPI, FP2\}$  is less than 0.0006. The failure probability of failure type  $\{FS2\}$  is required to be

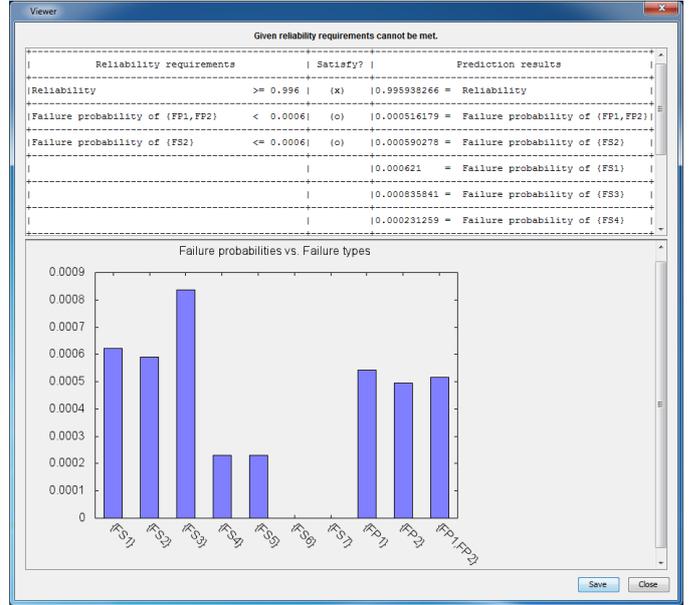


Fig. 11. Hybrid presentation of the prediction results of the reporting service.

less than or equal to 0.0006.

By following the process in Fig. 9, we obtained different presentations of the prediction results of the reporting service's variant. Fig. 11 shows the hybrid (including textual, tabular, and graphical) presentation as an example. All the prediction results are exactly the same as those of the former study. As the predicted reliability of the reporting service's variant is  $0.995938266 < 0.996$ , the first requirement cannot be satisfied, leading to a conclusion that "Given reliability requirements cannot be met.".  $\{FS1\}$  and  $\{FS3\}$  are the most frequent failure types, therefore, in order to achieve the first requirement, the service composer may recognize the need to introduce fault tolerance mechanisms for these failure types or to exchange services  $S_1$ ,  $S_3$ , or  $S_8$ , which cause these failure types, with more reliable candidates.

All of the above results give evidence that RMPI-SerComp's prediction workflow, along with the plug-ins, gives accurate presentations of prediction results in this case.

## VII. ASSUMPTIONS AND LIMITATIONS

The reliability prediction integrated into service composition frameworks via utilizing RMPI-SerComp is entirely powered by RMPI. Therefore, it shares the same set of assumptions and limitations with RMPI. For example, RMPI assumed that reliability-related behaviors of parallel branches of a parallel structure are independent, hence if the composition workflow of a composed service includes a parallel structure where the assumption is violated, then the integrated reliability prediction could lead to incorrect prediction results. We refer to our former work [9] for a detailed discussion of the assumptions and limitations of RMPI as well as techniques to relax them.

## VIII. CONCLUSION

In this paper, we presented RMPI-SerComp to offer enhanced integration support into service composition frameworks for RMPI, our recent model-based reliability prediction technique. With RMPI-SerComp, service composition frameworks can realize our recommended reliability-aware service composition process, enabling a systematic consideration of service reliability during composing without requiring specialist knowledge of the underlying prediction method. RMPI-SerComp offers a public API and a plug-in architecture for flexible and reusable integrations into service composition frameworks. Via a case study, we demonstrated the applicability of our approach by utilizing RMPI-SerComp to enhance a service composition framework with the reliability prediction capability according to our recommended process.

We plan to continue enhancing RMPI-SerComp, to include pre-installed translator and generator plug-ins for RMPI-SerComp, and to validate further our approach. These extensions will further increase the applicability of our approach.

## ACKNOWLEDGMENTS

We are grateful to François Bonnet, JAIST, for his invaluable comments that greatly helped improve this manuscript.

## REFERENCES

- [1] T. Erl, P. Chelliah, C. Gee, J. Kress, B. Maier, H. Normann, L. Shuster, B. Trops, C. Utschig, P. Wik *et al.*, *Next Generation SOA: A Concise Introduction to Service Technology & Service-Oriented*. Pearson Education, 2014.
- [2] Microsoft. (2013) BizTalk Server. [Online]. Available: <http://www.microsoft.com/biztalk/>
- [3] Oracle. (2014) Oracle BPM. [Online]. Available: <http://www.oracle.com/us/technologies/bpm/>
- [4] JBoss. (2015) jBPM. [Online]. Available: <http://jboss.org/jbpm/>
- [5] V. Grassi. "Architecture-based reliability prediction for service-oriented computing," in *Architecting Dependable Systems III*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, vol. 3549, pp. 279–299.
- [6] V. Cortellessa and V. Grassi, "Reliability modeling and analysis of service-oriented architectures," in *Test and Analysis of Web Services*. Springer Berlin Heidelberg, 2007, pp. 339–362.
- [7] Z. Zheng and M. R. Lyu, "Collaborative reliability prediction of service-oriented systems," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*. Cape Town, South Africa: ACM, 2010, pp. 35–44.
- [8] B. Klatt, F. Brosch, Z. Durdik, and C. Rathfelder, "Quality prediction in service composition frameworks," in *Service-Oriented Computing - ICSOC 2011 Workshops*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7221, pp. 131–146.
- [9] T.-T. Pham, X. Défago, and Q.-T. Huynh, "Reliability prediction for component-based software systems: Dealing with concurrent and propagating errors," *Science of Computer Programming*, vol. 97, Part 4, no. 0, pp. 426 – 457, 2015.
- [10] M. Edwards. (2011) Service component architecture. [Online]. Available: <http://www.oasis-open.org/sca/>
- [11] R. Cortez and A. Vazhenin, "Virtual model-view-controller design pattern: Extended mvc for service oriented architecture," *IEEE Trans. on Electronics, Sec. TEEC, C, Info. and Sys.*, vol. 10, no. 4, 2015.
- [12] A. Strunk, "QoS-Aware Service Composition: A Survey," in *IEEE 8th European Conference on Web Services (ECOWS)*, 2010, pp. 67–74.
- [13] K. Goseva-Popstojanova and K. S. Trivedi, "Architecture-based approaches to software reliability prediction," *Computers and Mathematics with Applications*, vol. 46, no. 7, pp. 1023–1036, 2003.
- [14] S. S. Gokhale, "Architecture-based software reliability analysis: Overview and limitations," *IEEE Trans. Dependable Secur. Comput.*, vol. 4, no. 1, pp. 32–40, 2007.
- [15] A. Immonen and E. Niemelä, "Survey of reliability and availability prediction methods from the viewpoint of software architecture," *Software and Systems Modeling*, vol. 7, no. 1, pp. 49–65, 2008.
- [16] M. Lyu, *Handbook of software reliability engineering*. IEEE Computer Society Press, 1996.
- [17] R. C. Cheung, "A user-oriented software reliability model," *IEEE Trans. Softw. Eng.*, vol. 6, no. 2, pp. 118–125, 1980.
- [18] V. Cortellessa and V. Grassi, "A modeling approach to analyze the impact of error propagation on reliability of component-based systems," in *CBSE*, 2007, pp. 140–156.
- [19] A. Filieri, C. Ghezzi, V. Grassi, and R. Mirandola, "Reliability analysis of component-based systems with multiple failure modes," in *Proceedings of the 13th international conference on Component-Based Software Engineering*, ser. CBSE'10, 2010, pp. 1–20.
- [20] W.-L. Wang, D. Pan, and M.-H. Chen, "Architecture-based software reliability modeling," *J. Syst. Softw.*, vol. 79, no. 1, pp. 132–146, 2006.
- [21] V. S. Sharma and K. S. Trivedi, "Reliability and performance of component based software systems with restarts, retries, reboots and repairs," in *Proceedings of the 17th International Symposium on Software Reliability Engineering*. IEEE Computer Society, 2006, pp. 299–310.
- [22] —, "Quantifying software performance, reliability and security: An architecture-based approach," *J. Syst. Softw.*, vol. 80, no. 4, pp. 493–509, 2007.
- [23] M. W. Lipton and S. S. Gokhale, "Heuristic component placement for maximizing software reliability," in *Recent Advances in Reliability and Quality in Design*, ser. Springer Series in Reliability Engineering. Springer London, 2008, pp. 309–330.
- [24] R. H. Reussner, H. W. Schmidt, and I. H. Poernomo, "Reliability prediction for component-based software architectures," *J. Syst. Softw.*, vol. 66, no. 3, pp. 241–252, 2003.
- [25] F. Brosch, H. Koziolok, B. Buhnova, and R. Reussner, "Architecture-based reliability prediction with the Palladio Component Model," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1319–1339, 2012.
- [26] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early prediction of software component reliability," in *Proceedings of the 30th international conference on Software engineering*. Leipzig, Germany: ACM, 2008, pp. 111–120.
- [27] S. Yacoub, B. Cukic, and H. H. Ammar, "A scenario-based reliability analysis approach for component-based software," *IEEE Trans. on Reliability*, vol. 53, pp. 465–480, 2004.
- [28] V. Cortellessa, H. Singh, and B. Cukic, "Early reliability assessment of UML based software models," in *Proceedings of the 3rd international workshop on Software and performance*. Rome, Italy: ACM, 2002, pp. 302–309.
- [29] P. Popic, D. Desovski, W. Abdelmoez, and B. Cukic, "Error propagation in the reliability analysis of component based systems," in *16th IEEE International Symposium on Software Reliability Engineering (ISSRE 2005)*, 2005, pp. 53–62.
- [30] K. Goseva-Popstojanova, A. Hassan, A. Guedem, W. Abdelmoez, D. Nassar, H. Ammar, and A. Mili, "Architectural-level risk analysis using UML," *IEEE Transactions on Software Engineering*, vol. 29, no. 10, pp. 946 – 960, 2003.
- [31] G. Rodrigues, D. Rosenblum, and S. Uchitel, "Using scenarios to predict the reliability of concurrent component-based software systems," in *Proceedings of the 8th international conference on Fundamental Approaches to Software Engineering*. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 111–126.
- [32] V. Grassi, R. Mirandola, and A. Sabetta, "Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach," *Journal of Systems and Software*, vol. 80, no. 4, pp. 528–558, 2007.
- [33] T.-T. Pham, F. Bonnet, and X. Défago, "Reliability prediction for component-based software systems with architectural-level fault tolerance mechanisms (Extended version)," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 5, no. 1, pp. 4–36, 2014.
- [34] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, 2004.
- [35] (2015) Reliability modeling, prediction, and improvements. [Online]. Available: <http://rmpi.codeplex.com/>