Towards a Formal Framework for Hybrid Analysis of Composite Web Services

May Haydar
Computer Science Department /DIRO
Fahad Bin Sultan University /Université de Montréal
Tabuk, Saudi Arabia /Montreal, QC
mhaidar@fbsu.edu.sa

Hesham Hallal
Electrical Engineering Department
Fahad Bin Sultan University
Tabuk, KSA
hhallal@fbsu.edu.sa

Abstract—In this work, we propose to develop an integrated formal framework where both static and dynamic analysis techniques complement each other in enhancing the verification process of an existing web services based application. The proposed framework consists of the following main components. The first component is a Library of Property Patterns which we intend to build on existing work [2, 14] and compile a library and a classification of web services properties (patterns and antipatterns [13]). These would include BPEL4WS and WISCI requirements in the form of property patterns which can be instantiated in different contexts and for different purposes like verifying correctness, security, and performance related issues. The property library will be based on an easy to use template that depicts mainly the type, formal model, and example of a property. The second component is the development of Static Analysis Techniques that include direct code inspection, abstraction based techniques, and model based techniques. The third component is the development of dynamic analysis techniques that include extracting behavioral models of applications from observed executions and verifying them (mainly using model checking) against behavioral properties. These properties specify defects that cannot be detected using static analysis techniques. A set of tools (prototype) are to be implemented to realize the proposed approaches for static analysis, modeling, and dynamic verification of the applications under test.

Static Analysis, Dynamic Analysis, Property Patterns, Web Services

I. INTRODUCTION

Businesses are increasingly adopting service orientation to shape the architecture of their enterprise solutions and to increase the efficiency of their software applications. At the foundation of this ever more popular paradigm, web services are heavily used to enhance decentralization and cross platform and language portability. The power of services resides mainly in the high degree of dynamism and flexibility they exhibit throughout their lifecycle: publication, discovery, and binding are all dynamic activities that make a service an evolving entity capable of adapting to continuously changing and new requirements. In addition, compositions of services, which can also be dynamic, have added to the power of services in building larger enterprise solutions for heterogeneous businesses. However, the fast paced growth of service implementation and deployment in various contexts has resulted in a growing gap between the development and verification of service based applications. On one hand, static analysis techniques [1, 13] remain insufficient to detect behavioral flaws and defects that are exhibited only when services, especially composite ones, are executed. In particular, such techniques face two major problems: difficulty of generating executable models that can be used in the analysis, and limited coverage of defects that are exhibited only during runtime, e.g., concurrency incurred problems. On the other hand, dynamic and runtime techniques, which depend mainly on monitoring, can only claim to detect errors and flaws in the observable behavior of a service, or a dynamic composition of services. In the meantime, formal methods have become a viable option to automate the verification process and increase its efficiency in modeling, testing, and error detection. Formal verification techniques are currently used in several domains including communications systems, software and program analysis [13], and web based applications [2, 14].
II. Formal Framework

We develop an integrated formal framework as illustrated in Figure 1, where both static and dynamic analysis techniques complement each other in enhancing the verification process of an existing web services based application. The proposed framework consists of the following main components.

A. Library of Property Patterns

Patterns have long been used in the development of software applications, and service oriented architectures as well, since they introduce clever and insightful ways to solve common problems. Along with patterns, which are intended to facilitate the design and development processes, the term antipattern is defined. An antipattern is simply a solution to a problem that does not work correctly.

Following the definition, efforts exist to document antipatterns in catalogs (similar to design patterns) so that they can be avoided. In the proposed framework, we intend to build on existing work [7,8,13,16,17] and compile a library and a classification of web services properties (patterns and antipatterns). The classification of properties will be hierarchical: static/dynamic, correctness/functional, style/performance, etc. Such classifications should help developers identify the antipatterns to better avoid them, and testers detect them in the application using the appropriate techniques. On the other hand, documented properties, which would include BPEL4WS and WISCI requirements in the form of property patterns, can be instantiated in different contexts and for different purposes like verifying correctness, security, and performance related issues. The property library will be based on an easy to use template that depicts mainly the type, formal model, and example of a property.

For example, in our previous work [16,17], we have defined a pattern template and identified 119 patterns and property specification for the verification Web applications (WAs). Figure 2 is an example of such patterns. Each pattern is specified in LTL which makes it easy to use in model checking.

B. Static Analysis Techniques

In general, static techniques for software, mainly based on analysis of (compiled) code (existing specifications or textual descriptions), are independent of specific input data sets or individual execution paths. They are usually classified into:

1. Direct code inspection, where suspicious code segments are directly identified in the code (through linear scanning for example).
2. Abstraction based techniques, where code representations (class diagrams, call graphs, etc.) are used to match the exhibition of certain predefined patterns (or antipatterns).
3. Model based techniques, where an executable model (often formal) is extracted from the code and verified against predefined properties using techniques like model checking or theorem proving.

![Figure 1. Formal Framework for Service Composition Analysis](image-url)
Figure 2. Web Specification Pattern

In the case of web services based applications, such techniques would be applied to the available documents containing the descriptions of individual and composite services. However, some complex faults cannot be detected with these approaches or only at a high cost (like deadlocks). Another disadvantage is false warnings (mainly false positives) that can be produced as results. This justifies the need for the third component, a set of dynamic analysis techniques.

C. Dynamic Analysis Techniques

These techniques have emerged as complementary to static analysis techniques, especially when concurrency and large architectural structures of applications make the latter inefficient and rather incomplete. Dynamic analysis techniques do not necessarily rely on existing specifications or textual descriptions of the applications under test. Instead, they are applied to executable behavioral models that are derived from the application’s observed executions (traces or logfiles). This solution is particularly efficient in the case of web services based applications, often characterized by their readiness to compose web services, especially dynamically. Although many standards have been introduced to address the problem of web service composition, including BPEL4WS (Business Process Execution Language for Web Services) and WSCI (Web Service Choreography Interface), they address mainly the description and execution of workflow specifications for web service compositions. Yet, they are not sufficient to support automated verification techniques based on static analysis. The proposed techniques include extracting behavioral models of applications from observed executions and verifying them (mainly using model checking) against behavioral properties specifying defects that cannot be detected using static analysis techniques. The known techniques in the field include:

1. offline (postmortem) techniques, where recorded executions of an application are stored and later used in modeling and verifying the application under test.
2. Online (runtime) techniques, where an application under test is analyzed as the executions are generated.

In our previous work [14,16], we designed a framework for formal modeling and verification of web applications. We intercepted http requests/responses that depict the behavior of web applications and extracted communicating automata models translated into Promela. We verified properties of WAs using Spin model checker. Results were promising and properties of concurrent behavior were verified, which could not be verified using other methods. Concurrent behavior of WAs represented behavior of WAs with multiple displays (windows/frames).

We use a similar approach for model extractions from behavioral executions of composite Web services. Collected traces are analyzed and abstracted as communicating automata models. Each automata depicts the behavior of one service where requests are modeled as events and responses as states. Events will be distinguished as local and common. Common events (rendezvous) represent service communications with each other.

III. IMPLEMENTATION

The implementation of the proposed framework includes the following main tasks:
1. Surveying the literature and common practices of various developers of web services based applications to compile a set of most frequently encountered properties (patterns and antipatterns).
2. Formulation of properties in specification languages that can be used in both static and dynamic analysis techniques.
3. Identifying proper static analysis techniques for each class of properties and evaluating their efficiency and robustness. In particular, this task includes identifying the proper abstractions, along with methods to extract them, to be used in detecting corresponding antipatterns in the code.
4. Record execution traces from the applications under test. This task includes studying the instrumentation based and interception based techniques.
5. Extracting models from monitored executions. This includes extracting models from completed traces and incremental models in the case of runtime analysis that can be used in known model checking tools.
6. Integrating the compiled library and developed tools in a user friendly toolset which masks the details of the underlying analysis techniques form the users and makes the dissemination of the produced framework easier.

The proposed framework is implemented using Spin model checker [18]. The automata models are represented using Promela language and the patterns/antipatterns are represented in LTL. We use the Java Eclipse environment for the toolset implementation. The complete toolset will include integrated components as follows:

- A library of compiled patterns/antipatterns translated in LTL.
- A monitoring tool that intercepts Web services communications and logs the intercepted behavior.
- An analysis module that analyzes and abstracts the intercepted communication (online or offline) and extracts a communicating automata model represented in XML.
- A model translator that translates the XML extracted model into Promela, the modeling language of Spin.
- A verification module that uses the Spin Model checker to verify patterns (from the library) on the extracted Web services automata model.
- A graphical user interface that coordinates the user command on the toolset.

Figure 3 illustrates our initial toolset prototype for the dynamic modeling of Web services.

![Figure 3. Prototype Tool for Web Services Monitoring and Modeling](image-url)

IV. RELATED WORK

Run time verification of software applications has grown as a major field covering major activities related to the development of software. At the same time, webbed, and web service-based, applications have gained a lot of attention in many research activities both in academia and in the industry given the role such applications have in the shaping of today’s economy based on e-commerce and e-services.

Our related work that is closely connected to this new proposed work is published in [8,14,16,18]. We have implemented an integrated formal framework for run-time verification of web applications. Results were interesting and we were able to verify properties that could not be verified using other approaches.

Recently, a large body of research has been produced with a focus on formal modeling of web services based applications in order to induce automation in the analysis of the developed
applications against some predefined properties specified from the description and requirements texts. Derived models are often generated from textual descriptions of applications (BPEL, BPEL4WS, and WSCI), and can be used mainly to check static properties that relate to the structure and content of the application, usually described as a composition of services. Examples of such research include the work of Foster et al. [1,2], which models BPEL descriptions as Finite State Process models, which can be verified against properties that are mainly derived from design specifications written in UML notations like the Message Sequence Chart (MSC) or activity diagrams. Properties sought for verification include mostly semantic failures and difficulties in providing necessary compensation handling sequences that are tough to detect directly in common workflow languages like BPEL. Other attempts have been described in the literature as well including the work of Breugel and Koshkina [3, 4] who introduce the BPE-calculus to capture control flow in BPEL descriptions and programs. The service descriptions in the proposed language allow for checking against properties like dead path elimination and control cycles. The verification, mainly formal model checking, is performed in the toolset Concurrency Workbench (CWB). However, as discussed in Section 1, proposed verification approaches based mainly on the static analysis of an existing source code, where different types of models like EFA, Promela, and communicating FSMs [11, 12] are used, have their limitations and impracticalities. Consequently, more efforts are being spent on performing run-time verification of web service applications based on monitoring and model extraction. Also, [5] address the run-time monitoring of functional characteristics of composed Web services, as well as for individual services [6].

V. CONCLUSIONS

In this paper, we proposed an integrated formal framework for the analysis and verification of Web services composition. We propose a hybrid of both static and dynamic analysis techniques which complement each other. We also intend to develop a library of patterns and antipatterns of interesting specifications of web services. These specifications will be automatically translatable to a formal specification language namely LTL. Based on our previous experience and the initial results obtained in the use of our formal approach for run-time verification, we believe that results of this proposed work are very promising.

REFERENCES

International Workshop on Web Languages and Formal Methods, 2(151):89-105.ENTCS.


