A Formal Hybrid Analysis Technique for Composite Web Services Verification

MAY HAIDAR [1,2], HICHAM H. HALLAL [1]
[1] Computer Science Department / Department of Electrical Engineering
Fahad Bin Sultan University
P.O Box 15700 Tabuk, 71454
SAUDI ARABIA
[2] Département d'informatique et de la recherche operationnelle
Université de Montréal
Pavillon André-Aisenstadt, CP 6128 succ Centre-Ville, Montréal QC, H3C 3J7
CANADA
{mhaidar,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 three components. A Library of Property Patterns, which we intend to build on existing work and compile a library and a classification of web services properties (patterns and antipatterns). 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. We elaborate in this paper the formal approach used to extract an automata based model of a web service composition from execution traces that are observed and collected using a monitoring tool. We also outline the components of a prototype to realize the proposed approaches for static analysis, modeling, and dynamic verification of the applications under test. In this paper we present the initial implementation of the dynamic approach.

Key-Words: - Static Analysis, Dynamic Analysis, Property Patterns, Web Services, Model Checking, Automata Models

1 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.
Currently, formal methods have gained momentum as a reliable solution to automate the analysis of various systems. In particular, software development communities are increasingly adopting formal techniques to perform different development activities such as requirement definition and elucidation, modeling and model transformation, testing, and property verification [14,17].

For example, model checking has been used to verify various properties on models of systems. Model checking can be fully automatic and produce counterexamples that point to the violations when a model does not satisfy a given property. Yet, as is the case for most formal analysis methods, adoption of model checking tools remains relatively limited due mainly to problems like the lack of formal models, the inherent state space explosion problem, and the lack of proper justification for its use especially for classes of properties whose verification does not explore concurrent behavior of the models **Error! Reference source not found.Error! Reference source not found.**. Yet, for many distributed applications and properties, especially those specified in terms of events issued by concurrent processes, the need for model checking becomes clear and outweighs the doubts cast over its use. Formal verification techniques are currently used in several domains including communication systems, software and program analysis [13], and web based applications [2, 14].

In the case of composite Web Services, the reasoning about the use of model checking is similar. While analyzing simple web services does not necessarily require the use of model checking techniques, the use of model checking in the analysis of web services featuring underlying dynamically composite services is clearly needed and justified. The latter is specifically true for services whose composition is specified through WS-BPEL [20] (Web Services Business Process Execution Language) and WSCI [21] (Web Services Choreography Interface).

As to the lack of models, especially in the case of inaccessible code, analysis in general has been applied to the traces an application/system produces when it is used. For this, predefined properties are used to analyze the application under test using model checking, when needed, or less complex techniques like search based methods or even manual inspection.

In this paper, we propose to develop an integrated formal framework where both static and dynamic analysis techniques complement each other in enhancing the property testing process of an existing web services based application.

2 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.

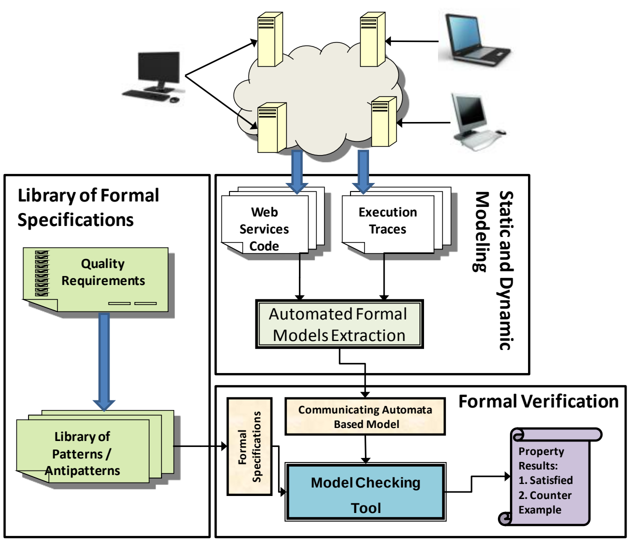

Library of Formal Specifications
Quality Requirements
Library of Patterns / Antipatterns
Web Services Code
Execution Traces
Static and Dynamic Modeling
Automated Formal Models Extraction
Formal Specifications
Communicating Automata Based Model
Formal Verification
Model Checking Tool
Property Results:
1. Satisfied
2. Counter Example

Figure 1. Formal Framework for Web Service Composition Analysis

2.1 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 as intended (in terms of correctness and/or efficiency).

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 classification 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 of Web applications (WAs). Figure 2 shows an example of such patterns. Each pattern is specified in Linear Temporal Logic (LTL), which makes it directly usable in many model checkers.

ID	FGS6
Pattern description	Banking information is entered no more than once before submitting form
Category	Functional – General – Security and Authentication
Page Attributes	*Banking_info*: Boolean identifying the presence of fields for banking information *Submit*: identification of page where form submit action exists
LTL Mapping	PrecedenceGlobally (($\neg(\, banking\_info)\,\mathrm{W}\,(banking\_info\,\mathrm{W}\,(\mathrm{G}\neg\,(banking\_info)))), submit)$
Comments	
Source	Newly introduced

Figure 2. Example of Web Applications Specification Patterns

2.2 Static Analysis Approach

In general, static analysis techniques for software, mainly targeting (compiled) code and/or 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).

In the case of web services based applications; static analysis techniques would be applied to the available documents containing the descriptions of individual and composite services. In doing this, we follow in the steps of the work in [13]; the main deviation being the customization of the antipattern library developed to handle mutlithreaded Java applications to the context of web services and web services compositions. In addition, the library will be extended to cover patterns/antipatterns like the one shown in Figure 2. However, some complex faults cannot be detected with static analysis approaches or only at a high cost (like deadlocks). Moreover, static analysis techniques are prone to producing significant numbers of false warnings (mainly false positives) while not being able to detect some behavioral errors like in the case of exception handling. This justifies the need for the third component, a set of dynamic analysis techniques.

2.3 Dynamic Analysis Approach

Dynamic Analysis 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). Such approach to analysis is particularly efficient in the case of web services based applications; often characterized by their readiness to compose web services, especially dynamically.

The communication between web services normally uses Internet protocols, such as HTTP, SMTP, and

FTP. However, all messages can be structured according to SOAP (Simple Object Access Protocol) which is a protocol specification for exchanging structured information in the implementation of web services. 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 WAs using the model checker Spin. We intercepted HTTP requests/responses that depict the behavior of web applications and extracted communicating automata models translated into Promela (the modeling language of Spin). The properties verified included properties of concurrent behavior of WAs featuring multiple displays (windows/frames).

We use a similar approach for model extractions from behavioral executions of composite Web services. The major difference is the availability of multiple traces recording the behavior of different services in the composition. The collected traces from all services are analyzed and abstracted as communicating automata models depicting the behavior of the respective services. Each automaton (inferred from a single trace) 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 represent communications among the services in a composition.

It is important to note that while the dynamic approach in this paper relies on model checking, models are derived from the observed behavior of the application. Thus, the approach could be seen as passive testing. Since results of verification could be compromised when a WSUT does not meet the assumptions described previously, this approach does not eliminate the need for traditional testing and should be considered as a complimentary activity rather than an alternative. For instance, the approach could be enhanced by additional testing of the application with model checking counterexamples, in order to verify whether properties are indeed violated. Also, behavioral models derived by this approach enable model based test generation **Error! Reference source not found.Error! Reference source not found.**.

In this paper, we propose a model checking based approach to the verification of web services composition whose source code is inaccessible against user defined properties. The model of the application under test is obtained from traces of the web services execution while properties of interest relate to both the business logic and ergonomics of the web services. More specifically, the proposed approach breaks down into the following main steps:

1. Modeling the Web services composition in a language acceptable by a chosen model checker. We use Spin **Error! Reference source not found.**, the open source model checker that is used in many research and industrial projects. As described earlier, we use the execution traces of the web services composition recorded using a relevant monitoring tool, e.g., a proxy server that is capable of intercepting HTTP and SOAP communications. The traces are then converted into a communicating automata model representing the behavior of all the components of the web services based application.
2. Specifying properties of interest. These properties can represent both desired and undesired behaviors of the web services. Properties will be mainly user defined and expressed in the property specification language of Spin, LTL.
3. Checking the obtained model against the given properties. To do so, Spin computes the composition of all the component automata in the derived model and builds a graph containing the global states of the application. The graph is then inspected against the language of a property for containment.

3 Automated Model Extraction of Web Services

The purpose of building a formal model for a web service under test (WSUT) is to verify whether the service composition exhibits certain predefined properties using model checking techniques. It is assumed in this paper that the properties specified in a temporal logic of a chosen model checker are composed of atomic propositions and for each SOAP/HTTP service request, the value of each proposition is uniquely determined by the content of the service response. These propositions refer to attributes that are user defined and have to be checked (and of course reflected in a model). Attributes can be of various types, for instance: a numerical type to count the occurrences of a certain element, a string type to denote the domain name of a response. To build a formal model of a web service composition whose source code is accessible, one may use abstraction techniques developed in software reverse engineering following a the *static*, white box approach **Error! Reference source not found.Error! Reference source not found.** as described in the previous section. However, the source code is not always available, or access to the code could breach copyrights or trade secrets (especially when verification is performed by a third party). Moreover, a web service composition can be written using different languages and even different paradigms which makes static analysis difficult to perform.

When the code is not available for modeling, one can build a formal model following a *dynamic*, black-box based approach, by executing the application and using only the observations of an external behavior of the service composition **Error! Reference source not found.** over a certain period of time. Verification of such models (resulting from finite trace of an application) is called run-time verification **Error! Reference source not found.Error! Reference source not found.**. In case of web services that rely on the SOAP or HTTP protocol considered in this work, an observable behavior consists of requests and responses, assuming that the flow of requests and responses between a client side and a server in the WSUT is observable. One possible way of achieving this is to use a proxy server **Error! Reference source not found.**. A proxy server monitors the traffic between the client and the server and records it in proxy logs. The proxy logs, i.e., traces, contain the requests for composing services and the responses to these requests.

In the next section, we present our approach to derive automata based models from traces of web services.

3.1 Modeling Approach

Figure 3 shows the workflow of the proposed approach. The main components are:

- A Monitoring module. It intercepts SOAP/HTTP requests and responses during the navigation of the WSUT performed by the user/crawler.
- An analysis module. It takes the intercepted traces as input and generates an automata model in XML/Promela. This module is realized as a prototype tool which is described in Section 4.
- A model checking module, in this case Spin. It verifies user defined properties against the generated model and produces a counterexample for each violated property.

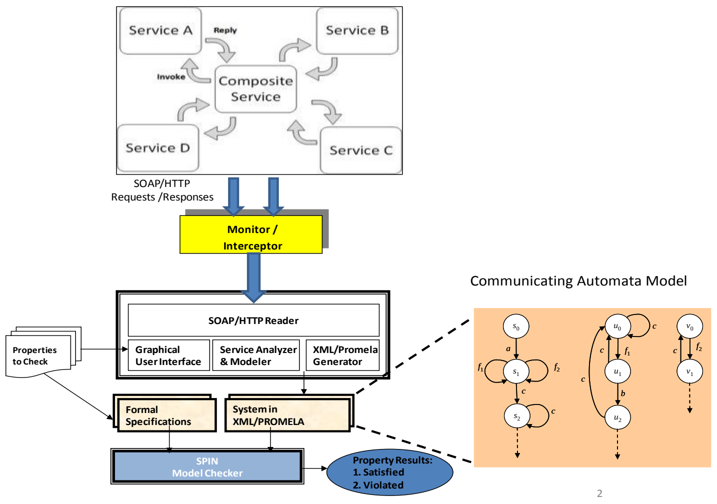


Figure 3. Workflow of the approach

With this approach, a behavior of a WSUT, called an *execution session*, aka Request/Response Sequence (*RRS*), is interpreted as a possible sequence of web services responses intermittent with the corresponding requests. Usually, many of these requests are triggered by the user's actions (clicking links, submitting forms), while others can be triggered by the service itself.

3.2 Execution Session as Communicating Automata

Here, the method for modeling an observed execution session by a system of communicating automata is described in general. Further development of the detailed model will be attempted in the future. Given the execution session, first local

execution sessions that correspond to the behaviors of the composition web services of the WSUT are determined, each of which is modeled by an automaton.
Automata communicate synchronously by rendezvous, executing common (rendezvous) actions. Such communication is formalized by the parallel composition operator on automata. Formally, two communicating automata $A_1 = < S_1, s_{01}, \Sigma_1, T_1 >$ and $A_2 = < S_2, s_{02}, \Sigma_2, T_2 >$ are composed using the $\|$ operator. The resulting automaton, denoted $A_1 \| A_2$, is a tuple $< S, s_0, \Sigma, T >$, where $s_0 = (s_{01}, s_{02})$ and $s_0 \in S$; $\Sigma = \Sigma_1 \cup \Sigma_2$; and $S \subseteq S_1 \times S_2$ and T are the smallest sets which satisfy the following rules:

- If $(s_1, e, s'_1) \in T_1$, $e \notin \Sigma_2$, and $(s_1, s_2) \in S$, then $(s'_1, s_2) \in S$, and $((s_1, s_2), e, (s'_1, s_2)) \in T$.
- If $(s_2, e, s'_2) \in T_2$, $e \notin \Sigma_1$, and $(s_1, s_2) \in S$, then $(s_1, s'_2) \in S$, and $((s_1, s_2), e, (s_1, s'_2)) \in T$.
- If $(s_1, e, s'_1) \in T_1$, $(s_2, e, s'_2) \in T_2$, and $(s_1, s_2) \in S$, then $(s'_1, s'_2) \in S$, and $((s_1, s_2), e, (s'_1, s'_2)) \in T$.

The composition is associative and can be applied to finitely many automata.

Local Execution Sessions. An execution session represents the behavior of communicating services denoted $o_1, o_2, \ldots, o_k$, where o_1 corresponds to the main composing service and k is the number of communicating services. Given an execution session, the number of communicating entities k and their relationship are determined and a procedure is used to partition the browsing session into local execution session, denoted $(RRS_1, \ldots, RRS_k)$.

4 Implementation of the Approach

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.

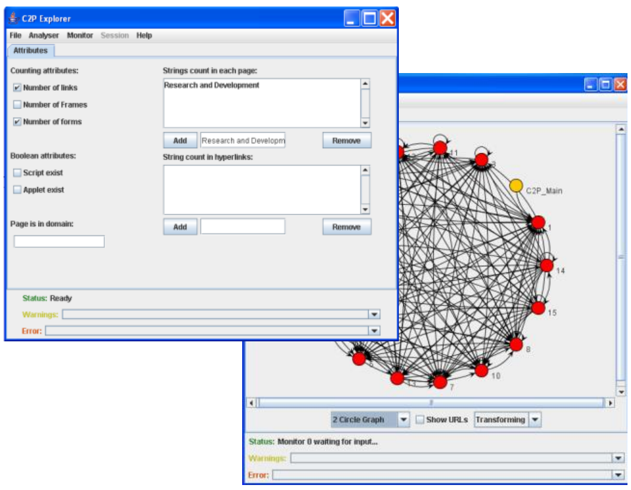
The proposed framework is implemented using Spin model checker [18]. The automata models are represented using Promela language and the patterns/antipatterns are represented in LTL. We use the Java Eclipse environment for the toolset implementation. The complete toolset will include integrated components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

a. *A library* of compiled patterns/antipatterns translated in LTL.
b. *Execution interception and monitoring*: the tool intercepts requests and responses of a web services composition using an open source proxy[1]. The monitoring module can operate in two modes: online and offline mode. In online mode, the monitor intercepts the executions and feeds them to the analysis module. In the offline mode, the monitor registers an execution trace in a log file.
c. *Property based attribute selection: thro*ugh the graphical user interface of the tool, a number of predefined attributes that characterize web services are provided. The user selected attributes are evaluated in each web service and are reflected in the automata model.
d. *Analyzing execution traces and model generation*: the tool parses and analyzes the execution traces and evaluates the us*er defined attrib*utes in each visited page. An internal data structure of the automata model of the web services composition is built. The model can be generated either in Promela language or XML-Promela.

[1] *SOLEX, Web Application Testing with Eclipse.* http://solex.sourceforge.net/

e. *Automata model visualization and statistical data*: the tool has a model visualization feature. The built model of a web services composition can be visualized in two different graphical modes as well as one textual model. In the graphical mode, which is based on existing Java graph libraries, both single automaton, and communicating automata models are visualized, which can be manipulated by the user. For instance, the user can zoom in/out, pick displayed states and drag them, visualize the content of each state, and optionally show/hide transition labels. Also, the tool provides numerical data about the model, namely the number of processes (automata), total number of states, total number of transitions.

Figure 4 illustrates our initial toolset prototype for the dynamic modeling of Web services.



Counting attributes:
[✓] Number of links
[ ] Number of Frames
[✓] Number of forms

Strings count in each page:
Research and Development

Boolean attributes:
[ ] Script exist
[ ] Applet exist

String count in hyperlinks:

Page is in domain:

Status: Ready

Status: Monitor 0 waiting for input...

Figure 4. Prototype Tool for Web Services Monitoring and Modelling

5 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].

6 Conclusion

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. We presented the formal framework for run-time verification of Web services composition as well as the extracted automata model.

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 promising.

References

[1] Foster, H. (2008). Tool Support for Safety Analysis of Service Composition and Deployment Models. Proceedings of *the 2008 IEEE International Conference on Web Services*, pp. 716-723. IEEE Computer Society.

[2] Foster, H., Uchitel, S., Magee, J., & Kramer, J. (2003). Model-based Verification of Web Service Compositions. Proc. of 18th IEEE International Conference on Automated Software Engineering, pp. 152-161. Montreal, Canada.

[3] Koshkina, M., & van Breugel, F. (2004). Modeling and verifying web service orchestration by means of the concurrency workbench. SIGSOFT Software Engineering Notes, 29(5):1-10. ACM.

[4] Van Breugel, F., & Koshkina, M. (2005). Dead-Path-Elimination in BPEL4WS. Proceedings of the 5th International Conference on Application of Concurrency to System Design, pp. 192-201. IEEE Computer Society.

[5] Kallel, S., Char_, A., Dinkelaker, T., Mezini, M., Jmaiel, M.: Specifying and Monitoring Temporal Properties in Web services Compositions. Proceedings of the 7th IEEE European Conference on Web Services (ECOWS). (2009).

[6] Simmonds, J., Gan, Y., Chechik, M., Nejati, S., O'Farrell, B., Litani, E., Waterhouse, J.: Runtime Monitoring of Web Service Conversations. IEEE Transactions on Services Computing. 99, 223-244 (2009).

[7] May Haydar, Sergiy Boroday, Alexandre Petrenko, and Houari Sahraoui . "Properties and Scopes in Web Model Checking". In Proc. of 20th IEEE/ACM International Conference on Automated Software Engineering (ASE 05). Long Beach, California, USA, November 2005.

[8] May Haydar, Sergiy Boroday, Alexandre Petrenko, and Houari Sahraoui. "Propositional Scopes in Lenear Temporal Logic". In Proc. of 5th International Conference on New Technologies of Distributed Systems (NOTERE 05). Gatineau, Quebec, Canada, August 2005.

[9] Dwyer M, Avrunin GS, Corbett JC. Patterns in Property Specifications for Finite-state Verification. 21st Int. Conference on Software Engineering, May, 1999.

[10] X. Fu et al, Analysis of interacting BPEL web services. 13th Int. World Wide Web Conference, 2004.

[11] Nakajima, S. (2006, May). Model-Checking Behavioral Specification of BPEL Applications. Proceedings of the International Workshop on Web Languages and Formal Methods, 2(151):89-105.ENTCS.

[12] Fu, X., Bultan, T., & Su, J. (2004). Analysis of interacting BPEL Web Services. Proceedings of the 13th International World Wide Web Conference, pp. 621-630. ACM Press.

[13] H. H. Hallal, E. Alikacem, W. P. Tunney, S. Boroday, A. Petrenko,(2004) "Antipattern-Based Detection of Deficiencies in Java Multithreaded Software," qsic, pp.258-267, Quality Software, Fourth International Conference on (QSIC'04).

[14] Haydar, M., Petrenko, A. and Sahraoui, H. (2004) "Formal Verification of Web Applications Modeled by Communicating Automata" In Proceedings of 24th IFIP WG 6.1 IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE 2004), pp. 115-132. Madrid, Spain. [LNCS, vol. 3235]

[15] Boroday, S., Petrenko, A., Sing, J. and Hallal, H. (2005) "Dynamic Analysis of Java Applications for MultiThreaded Antipatterns" In Proceedings of the Third International Workshops on Dynamics Analysis (WODA 2005). St-Louis, MI, USA.

[16] May Haydar. *A Formal Framework for Run-Time Verification of Web Applications: An Approach Supported by Scope Extended Linear Temporal Logic*. VDM Verlag, Germany, September 2009. ISBN: 978-3-639-18943-8.

[17] May Haydar, Houari Sahraoui, and Alexandre Petrenko. "Specification Patterns for Formal Web Verification". In Proc. of 8th International Conference on Web Engineering (ICWE 08). Yorktown Heights, New York, USA, July 2008.

[18] Gerarld Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. ISBN-10: 0321228626. Addison-Wesley, Sptember 2003.

[19] A. Andrews, J. Offutt J, R. Alexander, Testing Web Applications by Modeling with FSMs, Software Systems and Modeling, 4(3):326-345, July 2005.

[20] OASIS. (2007). *OASIS Web Services Business Process Execution Language*. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[21] Arkin, A., Askary, S., Fordin, S., & al. (2002). *Web Service Choreography Interface* (*WSCI*) *1.0*. Retrieved on April 10, 2005 from www.w3.org/TR/wsci.