# Integrating Formal Approaches and Simulation to Improve Reliability and Correctness of Web Services

George Eleftherakis[1] and Ognen Paunovski[2]

[1] City College, 13, Tsimiski str., 54624 Thessaloniki, Greece

[2] South East European Research Centre, 17, Mitropoleos str., 54624 Thessaloniki, Greece

**Abstract.** The emerging web service paradigm offers an innovative and practical platform for business to business collaboration and enterprise information systems integration. A methodology for modelling web service systems based on an incremental and iterative approach integrating formal techniques and simulation is presented. This disciplined approach focuses on improving the reliability and correctness of the system under development. Using X-machines as the core design technique it offers intuitive mapping of BPEL specification. At the same time it enforces continuous verification and testing of components throughout the process. Blending this formal approach with simulation it allows the informal verification of complex service compositions in cases where formal verification is impossible or impractical. The applicability of the methodology is practically demonstrated through a typical web service case study.

## 1 Introduction

The future interoperation between network applications will be based heavily on the concept of Web Services. Web Services (WS) as self-contained software components aim to provide seamless machine to machine interoperation in network applications. This new paradigm is of paramount importance for business to business collaboration and enterprise information systems integration. This is primarily due to the broad range of applicability and flexibility of the architecture as well as the opportunity to facilitate introduction of novel functionalities which can be achieved through service collaboration. This collaboration between the services creates a framework for combination of existing services in order to achieve a desired business process. Thus the elementary services may form more complex composite services.

However as the complexity of web services increases, there is a need to ensure that they behave correctly. Therefore the elementary services need to satisfy several criteria. First of all they need to meet the requirements and satisfy any necessary properties which are part of its design objectives. Additionally the implementation of the service should follow the design and be tested in order to prove its correctness. In this context it is argued that the use of formal methods can achieve (to some extend) this goal in all phases of system development, modelling, verification and testing [10]. However applying formal verification to composite web services is not possible in some cases due

Eleftherakis G. and Paunovski O. (2008).
Integrating Formal Approaches and Simulation to Improve Reliability and Correctness ofWeb Services.
In *Proceedings of the 6th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems*, pages 180-189
DOI: 10.5220/0001740701800189

to the complexity of the composition. Furthermore in some cases while formal verification may be possible it requires too much time and effort which makes it completely impractical.

This is in fact why an informal language is commonly used to specify compositions of web services. The Business Process Execution Language for Web Services (BPEL4WS or BPEL) [1] in recent years has become an industrial standard for specifying web service compositions. Nevertheless while BPEL is intuitive and easy to use it lacks the capability to ensure "correctness" of the design. A possible solution to this problem could be achieved by mapping a BPEL process specification to a language with formally defined semantics (e.g. Petri net, state machines, process algebras etc.). This approach has several advantages. By mapping a BPEL specification to a formal language, a formal semantics of the BPEL could be provided [12]. Also, all static and dynamic analysis techniques and tools developed for the formal method can be exploited in the context of BPEL processes improving the confidence of the correctness of the final product.

In this paper we use a formal method, namely X-machines and its extension Communicating X-machines, to capture a BPEL process specifications. The chosen formal method closely suits the needs of component-based development (like in the case of web services) while being practical at the same time. Furthermore we present a disciplined methodology offering iteratively incremental development of complex service compositions. The proposed methodology utilises formal modelling and verification to avoid any flaws in the early stages of the development of services together with a formal testing strategy to discover any undiscovered flaws in later stages. These formal techniques are coupled with informal verification steps provided through simulation (animation). The simulation is needed in order to informally verify complex models with dynamic communication which cannot be formally verified. However at the same time the animation of the model is a step which provides immediate feedback to the development team and facilitates effective communication of the formal experts and the people (users and/or developers) with no formal background. All these features make the proposed methodology practical. This way it makes the best use of the development effort to achieve highest confidence in the quality of the developed services.

In the following section several formal approaches used in web service development are discussed with the focus on the one used in this paper. The major idea and a brief description of the activities in the proposed methodology is elaborated in section 3. We demonstrate the approach taken to implementing each activity in the methodology with the same example throughout. Finally, a discussion on the methodology and further work to be carried out is presented.

## 2 Web Services and Formal Methods

There are several techniques and tools that are able to transform a process specified in BPEL into a formal model for the purpose of verification. In this step the formal model is usually transformed into a version of finite state machines (FSM) and automata [7], a version of Petri nets, process algebras [14] or other formal methods. The second step

is to represent the formal model (e.g. FSM) to an appropriate language for a model checker or any other tool offering verification for the model [11].

In this process the emphasis is on the verification of web service specification, however there is very little ongoing research on the testing of web services [3]. There are even fewer attempts to combine testing techniques with verification techniques, like the work described in [6], where the focus is on testing of composite web services. However there is a lack of methodologies integrating these formal techniques into a practical process.

The proposed methodology is addressing the issues discussed above through a disciplined process which utilises a formal approach (X-machines) and integrates many formal techniques with informal activities (simulation). In essence X-machines is a formal method that enhances the class of FSM by introducing memory and functions. An X-machine is defined by an input stream, an output stream, a set of values that describe its memory structure, a set of states, a state transition set and a set of functions. Labels in the transitions are functions which are triggered through an input symbol and a memory instance to produce an output symbol and a new memory instance. A deterministic X-machine [5] is an 8-tuple $X = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:

- $\Sigma$ and $\Gamma$ are the input and output alphabets respectively.
- $Q$ is the finite set of states.
- $M$ is the (possibly) infinite set called memory.
- $\Phi$, the *type* of the machine $X$, is a set of partial functions $\varphi$ that map an input and a memory state to an output and a possibly different memory state, $\varphi : \Sigma \times M \rightarrow \Gamma \times M$.
- $F$ is the next state partial function, $F : Q \times \Phi \rightarrow Q$, which given a state and a function from the type $\Phi$ determines the next state. $F$ is often described as a state transition diagram.
- $q_0$ and $m_0$ are the initial state and initial memory respectively.

X-machines can be applied in similar cases as Statecharts and other similar notations, such as SDL for example. However, X-machines have several significant advantages. First, they provide a mathematical modelling formalism for the system, which in turn allows X-machine specification to be model checked [2]. Thus, facilitating the verification of desired model properties. Moreover, X-machines offer a strategy to test the implementation against the model which guarantees to determine correctness if certain assumptions in the implementation hold [5].

In addition, communicating X-machines provide a notation allowing to define interaction and communication between individual X-machine models [8]. Functions can send messages to input streams of other X-machine components which are consumed by local functions. A Communicating X-machine System $Z$ as defined in [8] is a 2-tuple $Z = ((C_i)_{i=1,\ldots,n}, CR)$ where:

- $C_i$ is the $i$-th Communicating X-machine component, and
- $CR$ is a relation defining the communication among the components, $CR \subseteq C \times C$ and $C = \{C_1, \ldots, C_n\}$. A tuple $(C_i, C_k) \in CR$ denotes that the X-machine component $C_i$ can output a message to a corresponding input stream of X-machine component $C_k$ for any $i, k \in \{1, \ldots, n\}$, $i \neq k$.

A communicating X-machine model consists of several X-machine models that are able to interact by exchanging messages. The structure $CR$ defines a directed graph which statically determines the direction of messages between components. An X-machine component is defined as an X-machine in which the functions do not only read and write from/to their input and output streams respectively but also read and write from/to streams that are used to communicate with other X-machine components. More analytically, functions are of the form: $\varphi_i\ ((\sigma)_j, m) = ((\gamma)_k, m')$ where $(\sigma)_j$ means that input is provided by machine $C_j$ whereas $(\gamma)_k$ denotes an outgoing message to machine $C_k$. If $i = j$ and/or $i = k$, that means that machine $C_i$ reads from its standard input stream and/or writes to its standard output stream.

In practice, it is found that the development of a communicating system model can be based on a number of well-defined distinct steps that are described in detail in the following sections. To each of the steps a set of appropriate tools, such as an interchange description language, parser, animator, test set generator etc., is employed in order to make the methodology applicable in real cases [9].

## 3 Methodology

Communicating X-machines is viewed as a modelling method, where a complex system can be decomposed in small components (elementary services) modelled as simple X-machine models, thus model interacting component-based systems. The communication of all these components is specified separately in order to form the complete system as a communicating X-machine model which corresponds to a composite web service. This implies a modular bottom-up approach and supports an iterative gradual development. It also facilitates the reusability of existing X-machine models, making the management of the whole project more flexible and efficient. Thus achieving the completion of the entire model with lower cost and less development time.

The communicating X-machine method supports a disciplined modular development, allowing the developers to decompose the system under development into communicating components. We suggest that the development of a system model can be mapped into the following well-defined distinct actions that are graphically illustrated in figure 1:

1 Analyze the existing business process in order to determine the web service composition:
   - description of the tasks performed by individual web services,
   - description of the communication (interaction) between the web services,
   - description of the expected behaviour of the composed web service.
2 Develop a set of test scenarios (simulation conditions) which will validate the behaviour of the service composition. At the same time the set of properties that each of the services should satisfy need to be derived.
3 Develop X-machine models for each independent service in the composition.
4 Code the X-machine model into a language (XMDL) that facilitates the subsequent steps.

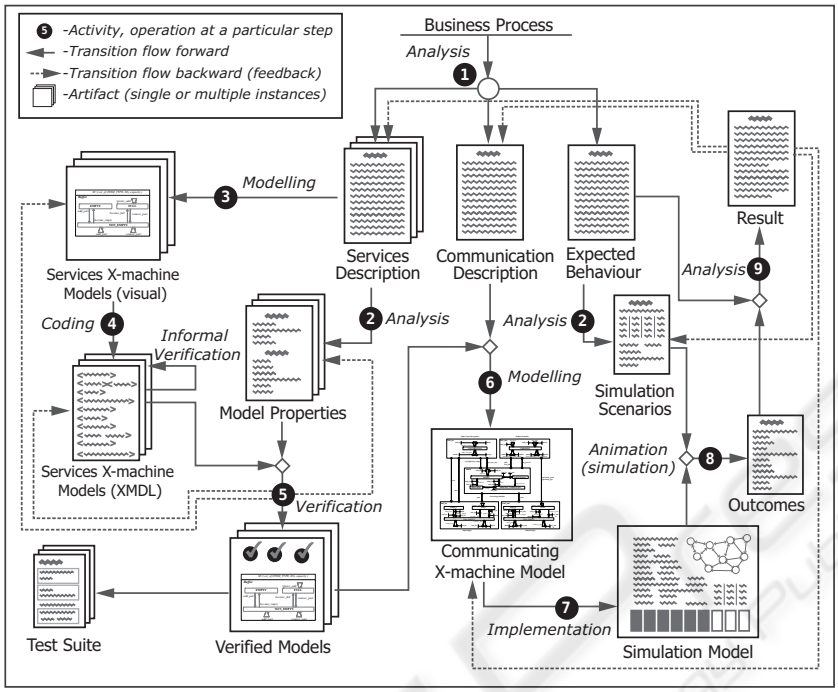


**Fig. 1.** The proposed development methodology.

- Simulate the X-machine model to rapidly verify (informally) the expected behaviour and communicate the model to the users to detect errors in the design in the early stages of the development.

5 Use the formal verification technique (model checking) for X-machine type models in order to increase the confidence that the proposed model has the desired characteristics.

6 Develop communicating X-machine model which describes the way in which individual services communicate and interact.

7 Implement the communicating X-machine model in a form which could be animated, used in a simulation study.

8 Execute the test scenarios (defined in step 2) through execution of simulation study of the implemented model.

9 Analyze the results of the simulation on order to determine whether the service composition behaved as intended.

The process described above can be used to refine the resulting model following an iterative process. Towards this end a set of appropriate tools has been developed and have been integrated under a suite (called X-System), to support modelling with X-machines [9]. X-System can be employed to facilitate the activities of the above methodology making it applicable in real cases. Coding of X-machine model is carried out using the X-machine Description Language (XMDL) notation which acts as

an interchange language for describing X-machine models and its corresponding tools (syntax and type checker, visual editor, compiler, animator) [9]. Through the animation, it is possible for the developers to informally verify that the model corresponds to the actual system under development. The animation can also be used to demonstrate the model to the end-users allowing them to identify any misconceptions regarding the user requirements. After that formal verification of X-machine models is achieved with the use of an automated tool, a model checker. Model checking of X-machine models is supported by $\mathcal{X}m$CTL . This technique enables the designer to verify the developed model against temporal logic $\mathcal{X}m$CTL formulas which express the properties that the system should have. Following the implementation of the service, the test-cases are automatically derived using the X-machine test case generator. This allows the use of the formal testing strategy to test the implementation and prove its correctness with respect to the X-machine model. Once the individual models are verified, the communication and interaction of the components can be established. This is done in XMDL-c notation and its corresponding tools. Using this framework the simulation scenarios are executed to derive informal validation of the expected behaviour.

In the section that follows a web services example is used as a vehicle of study illustrating the proposed methodology and its applicability to composite web services, explaining in practice how each activity is carried out.

## 4 Case Study

In order to demonstrate the modelling of web services using X-machines, we have applied our methodology on the Virtual Travel Agency (VTA) case study [7]. In the VTA problem domain the goal of the VTA service is to provide a flight and hotel booking for a user. Once the VTA receives a reservation request from a user, it contacts the flight service in order to get an offer for the available flights in the specified period and location. After that it contacts the hotel service for the available hotel offers. When it receives the offers the VTA contacts the user with the available offers awaiting its response. If the user accepts the offer the VTA informs the flight and hotel services in order to obtain the tickets and provides them to the user. Otherwise (if user rejects the offer) the VTA rejects the hotel and flight service offers. In the case when no offer is available, either for the flight or hotel, the VTA informs the user that there is no offer available for the requested time and location.

First of all the analysis of the description of the case study produces the three expected documents. A description of the identified services (i.e. flight, hotel and vta) and their expected behaviour, a description of the way these services communicate and finally the description of the expected system behaviour. Using these documents we construct the needed simulation scenarios and we specify each model's properties as temporal logic formulas. Then we start modelling each service, using the first document, as X-machine models.

In figure 2 the state transition diagram of the X-machine model of the flight service is depicted together with the BPEL description of the service. This figure clearly depicts the intuitive modelling of the service as an X-machine model and demonstrates the expressional power of the X-machine to model web services. The extra benefit using

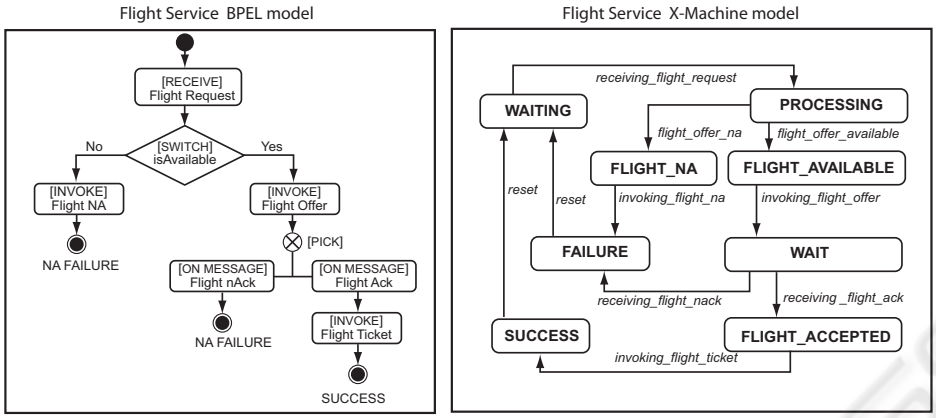


**Fig. 2.** Flight service in BPEL and as an X-machine model.

X-machines compared with other formal methods is that apart from the intuitive way to model the control (dynamic behaviour of the service) you can model the data (static part of the service). Using mathematical notation, the definition of the flight service is as follows:

- The set of inputs is $\Sigma$ = `BPEL_element` $\times$ `variable` where `BPEL_element` = { `invoke`, `receive`, `on_message`, `switch`, `reset` } and `variable` = `variable_name` $\times$ `value`, where `variable_name` is a basic type (could be any string) and `value` is either a natural number, or a boolean, or an abstract representation of date and time as time slots (`time_slots`). The set of outputs is $\Gamma$ ={`flight_requested`, `flight_available`, ...}.
- The set of states is $Q$ ={ `waiting`, `processing`, `flight_available`, `flight_na`, `failure`, `wait`, `flight_accepted`, `success`}.
- The memory holds a set of all the available flights and the flight that was requested. $M$ = (`flight_numbers` $\times$ `time_slots`, (`flight_numbers`,`time_slots`)), with `flight_numbers` representing all possible flight numbers.
- The type of the machine $\Phi$ is a set of the transition labels in figure 2.

Finally, the functions $\varphi \in \Phi$ of the X-machine need to be defined. The next activity is to code the model to XMDL. In order to demonstrate XMDL the function `flight_offer_available` is defined in XMDL as:

```
#fun flight_offer_available(((?bpel,(?F,?T))),(?av_flights,?fl))=
if ?bpel = switch and (?F,?T) belongs ?av_flights
then ( (flight_available), (?av_flights),(?F,?T)).
```

Using XMDL as the modelling language, X-System allows the animation of X-Machine models. This is an important activity that makes the methodology practical since it enhances the communication overall (developer to developer and developer to client) and provides early and cost effective feedback from the actual users although they have no experience in formal models.

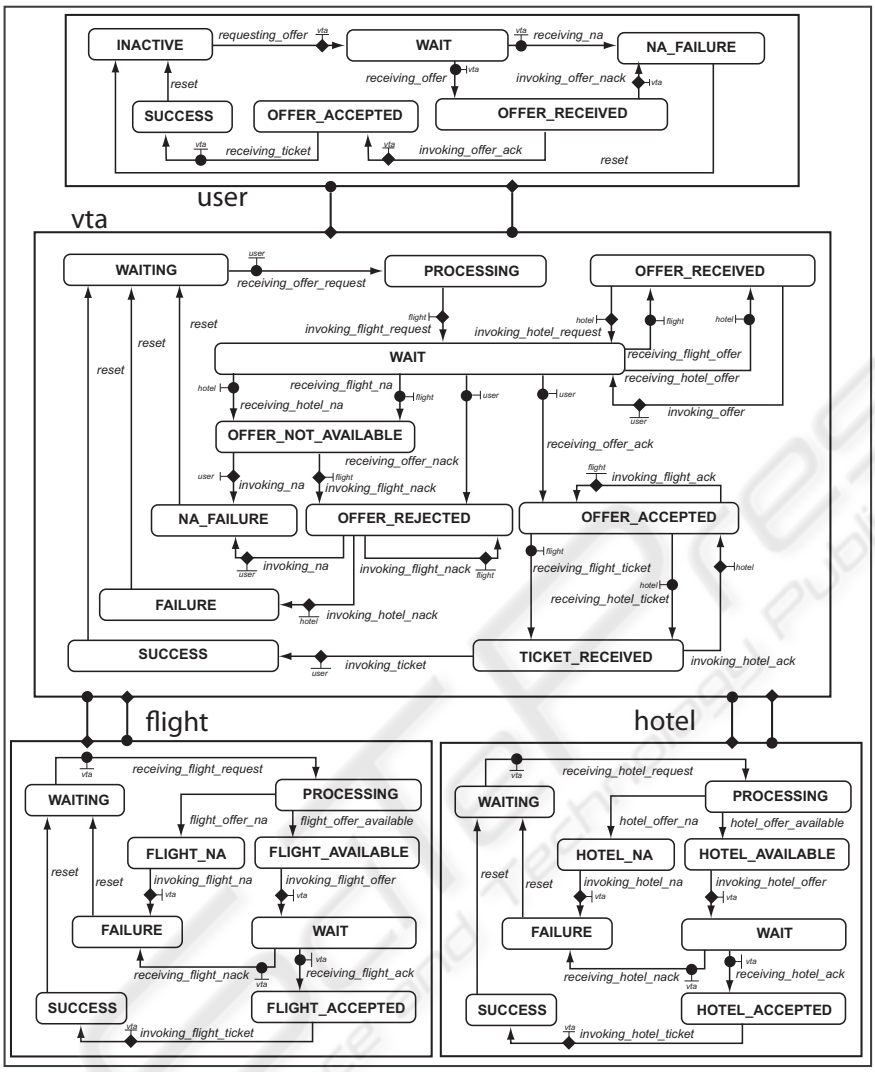


**Fig. 3.** The overall system as a communicating X-machine model.

However there are properties of the system that the designer would like to prove and not demonstrate their existence in the X-machine model. The formal verification technique for X-machine models enables the designer in the following activity to formally verify the developed model against temporal logic formulas that express the properties that the system should have.

The next step is to specify the communication between the X-machine models of the services and this way create the communicating X-machine model of the system under development. One more benefit is that it is possible to model the environment (i.e. the user) as an X-machine achieving to model the complete system. Graphically on the state

transition diagram we denote the acceptance of input by a stream other than the standard by a solid circle along with the name $C_j$ of the communicating X-machine component that sends it. Similarly, a solid diamond with the name $C_k$ denotes that output is sent to the $C_k$ communicating X-machine component. The complete model is depicted in figure 3 providing a flexible and modular design of the complete system where clearly the services of the system are represented together with the needed communication to create the expected system behaviour.

XMDL has been extended (XMDL-c) in order to code communicating components. XMDL-c provides syntax that facilitates the definition of the communicating functions. CommX-System is a tool created to support modelling and simulation of Communicating X-machines [8]. CommX-System takes as input all the XMDL files describing the services together with the XMDL-c file that describes the communication between them, and outputs an executable file corresponding to the overall system, offering a simulation model of it. The following activity is to execute the simulation scenarios and compare the outcome with the expected behaviour of the system. The results of this analysis will provide feedback to the developers indicating the "correctness" of the model. Taking this into account further iterations maybe needed to improve the model.

## 5 Conclusions and Further Work

We have presented a methodology for developing enterprise systems following a web service architecture using X-machines formal method. X-machines attracted many researchers interest over the last fifteen years [4] mainly because of the intuitiveness in modelling reactive systems and the additional features they provide in terms of testing and verification. The methodology and its accompanying tools impose an incremental bottom-up practical development.

The proposed methodology following a holistic approach, integrates formal techniques focusing on the improvement of reliability and correctness of composite web services. It offers a disciplined approach exhibiting the following key features: (a) It is component based and architecture centric. (b) It uses a formal method (X-machines) as the core design tool that offers a diagrammatic notation and tool support. This formal model offers an intuitive mapping of BPEL notation to a formal one. (c) It enforces continuous verification and testing of components throughout the process to achieve highest confidence in the quality of the developed components. (d) It employs simulation allowing the informal verification of complex service compositions in cases where formal verification is impossible or impractical. (e) Design is incremental and iterative via a prescribed sequence of design activities within a cyclic process.

The use of the proposed methodology on the case study demonstrated promising results. However, there are several issues that could be potentially improved. Future work will be concentrated on the automatic translation of BPEL specifications to X-machine models. In addition the model checking technique could be further extended in order to facilitate the formal verification of communicating X-machine models. Research is also conducted towards the establishment of a successful testing strategy for the communicating X-machine models, that is expected also to offer the ability to formally extract simulation scenarios in order to follow a disciplined approach in the informal verifica-

tion through simulation. Finally, on going research is addressing the issue of detecting unexpected emergent behaviours in automatic web services compositions [13].

## References


1. A. Arkin, S. Askary, B. Bloch, F. Curbera, Y. Goland, N. Kartha, C.K. Liu, V. Mehta, S. Thatte, P. Yendluri, A. Yiu, and A. Alves. Web services business process execution language, version 2.0, December 2005.
2. G. Eleftherakis. *Formal Verification of X-machine Models : Towards Formal Development of Computer-Based Systems*. PhD thesis, University of Sheffield, UK, 2003.
3. Lars Frantzen, Jan Tretmans, and René de Vries. Towards model-based testing of web services. In Antonia Bertolino and Andrea Polini, editors, *in Proceedings of International Workshop on Web Services Modeling and Testing (WS-MaTe2006)*, pages 67–82, Palermo, Sicily, Italy, June 9th 2006.
4. M. Holcombe. What are X-machines? *Formal Aspects of Computing*, 12(6):418–422, 2000.
5. M. Holcombe and F. Ipate. *Correct Systems: Building a Business Process Solution*. Springer Verlag, London, 1998.
6. Hai Huang, Wei-Tek Tsai, Raymond Paul, and Yinong Chen. Automated model checking and testing for composite web services. In *Proceedings of 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, page 300307, Seattle, WA, USA, May 2005.
7. R. Kazhamiakin and M. Pistore. Parametric communication model for the verification of BPEL4WS compositions. In M. Bravetti, L. Kloul, and G. Zavattaro, editors, *Proceedings of the 2nd International Workshop on Web Services and Formal Methods*, volume 3670 of *Lecture Notes in Computer Science*, pages 318–332, Versailles, France, September 2005.
8. P. Kefalas, G. Eleftherakis, and E. Kehris. Communicating X-machines: a practical approach for formal and modular specification of large systems. *Information and Software Technology*, 45(5):269–280, April 2003.
9. P. Kefalas, G. Eleftherakis, and A. Sotiriadou. Developing Tools for Formal Methods. In *9th Panhellenic Conference on Informatics*, pages 625–639, Thessaloniki, November 2003.
10. B. Meyer. The Grand Challenge of Trusted Components. In *25th International Conference on Software Engineering*, pages 660–667, Portland, Oregon, May 2003.
11. S. Nakajima. Lightweight formal analysis of web service flows. *Progress in Informatics*, 1(2):57–76, November 2005.
12. C. Ouyang, W.M.P. van der Aalst, S. Breutel, M. Dumas, A.H.M. ter Hofstede, and H.M.W. Verbeek. Formal semantics and analysis of control flow in ws-bpel. Technical Report Report BPM-05-15, BPM Center, 2005.
13. O. Paunovski, G. Eleftherakis, and A.J. Cowling. Framework for Exploring Emergence within Complex Systems. In *Proc. 2nd Annual SEERC Doctoral Conference*, July 2007.
14. G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. In *Proceedings of the IEEE International Conference on Web Services*, pages 43–51, San Diego, CA, USA, June 2004. IEEE.