An Agile Formal Development Methodology

George Eleftherakis[1] and Anthony J. Cowling[2]

[1] Computer Science Department
City Liberal Studies
Affiliated College of the University of Sheffield
13 Tsimiski Str., 54624 Thessaloniki, Greece
eleftherakis@city.academic.gr,
WWW home page: http://www.city.academic.gr
[2] Computer Science Department
University of Sheffield, UK
a.cowling@dcs.shef.ac.uk,
WWW home page: http://www.dcs.shef.ac.uk

Abstract. The demand for more complex but also more reliable and correct computer based systems on the one hand, and the fact that several changes in the user requirements through the development cycle on the other hand, leads to the need for more formal but also agile development methodologies. This report proposes $\mathcal{XF}un$, which is a development methodology that adopts the unified process, and proposes as the core modelling technique the X-machine formal method. This amalgamation creates a formal agile methodology aiming for the development of computerised systems that will be reliable and correct with respect to user requirements.

1 Introduction

The traditional approach to software development fails to cope with even small changes in the requirements at any stage after the analysis. It is generally accepted that the software development methodology which is mostly used in the industry is the waterfall model, which however exhibits an awkward behaviour in changes during the later stages of the development. This is the reason for the dramatic increases in time and cost of the development of software. The waterfall model is a linear sequential model which emphasizes the completion of one phase of the development before proceeding to the next one. The most common problems are the complexity overload, the delayed feedback, the requirement for unchanged requirements over the development time of the project thus increasing the cost of any change, and the delayed risk reduction. Methodologies like this are bureaucratic, they demand masses of documentation, and there is so much to do to follow the methodology that the whole pace of development slows down, hence they are often referred to as heavy methodologies.

The value of formal methods as a means of improvement in the development of reliable, high integrity, correct systems has long been accepted. But, it is also a common belief of people outside the formal methods community that

Proceedings of the 1st South-East European
Workshop on Formal Methods, SEEFM'03
November 2003, Thessaloniki, Greece

formal methods are difficult to understand and use. Furthermore formal specifications can be costly and time consuming. Taking into account that formal methods do not cope well with changes in the requirements which could result in major rework of the produced model, any attempt to combine them with the heavy software development methodologies will result in more rigid development methods. This will make the widespread use of formal methods in the software industry even more difficult because in commercial environments requirements change often throughout the development cycle.

In order to be useful to modelling of computer based systems a formal method should be able:

- to model both the data and the control of a system,
- to offer a practical, modular, disciplined way of modelling that will facilitate the modelling of large scale systems,
- to be intuitive, practical and effective towards implementation of the system, and
- to facilitate development of correct systems.

All the above are prominent characteristics of the X-machine. X-machine is a formal method introduced by Eilenberg [4]. Holcombe proposed X-machines as a basis for a possible specification language [9] which is capable of modelling both the data and the control of a system. With the development of a formal testing strategy [11], a formal verification technique [6] and a methodology of building communicating systems out of X-machine components [13], and with the added support of tools [14] and the proposal of a formal framework for the development of more reliable systems [5], most of the above mentioned requirements are met with emphasis in the development of correct systems.

Over the last years the so-called lightweight or agile methodologies have been introduced which attempt a useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff. Some of the proposed agile methodologies are [8]: Adaptive Software Development (ASD), Agile Software Process (ASP), Crystal, Dynamic System Development Method (DSDM), Extreme Programming (XP), Feature Driven Development (FDD), Unified Process (UP), SCRUM, etc.

X-machines have already been used together with an agile methodology, extreme programming, to provide valuable aid with the testing strategy they offer [10], but not to adopt it towards an integrated formal methodology. This paper argues that the UP is another appropriate lightweight software development process to adopt in order to combine it with X-machines towards a formal lightweight development methodology of computerised systems.

2 X-machines

X-machines employ a diagrammatic approach of modelling the control by extending the expressive power of FSM. Transitions between states are no longer

performed through simple input symbols but through the application of functions. In contrast to FSM, X-machines are capable of modelling non-trivial data structures by employing a memory, which is attached to the X-machine. Functions receive input symbols and memory values, and produce output while modifying the memory values (Fig. 1).

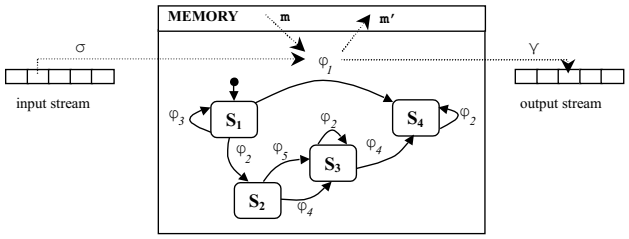

Fig. 1. An abstract X-machine model

The X-machine formalism as a modelling tool has many advantages:

- it is formal (making it suitable for mathematical analysis),
- it is rigorous,
- it is expressive,
- it provides unambiguous models,
- it is capable of capturing both static and dynamic system information,
- it is based on a fully general and formalised computational model, that could form the basis of a universal approach to the design of systems,
- it offers communicating X-machines, which means that it can support component based development, and
- it is supported by appropriate tools.

A particular class of X-machines is *stream X-machines* which is defined as an 8-tuple as follows [12]: $\mathcal{M} = (\Sigma, \Gamma, Q, M, \Phi, F, q_0, m_0)$ where:

- Σ, Γ is the input and output finite alphabet respectively,
- Q is the finite set of states,
- M is the (possibly) infinite set called memory,
- Φ is the type of the machine $\mathcal{M}$, a finite set of partial functions ϕ that map an input and a memory state to an output and a new memory state, $\phi : \Sigma \times M \rightarrow \Gamma \times M$
- F is the next state partial function that given a state and a function from the type Φ, denotes the next state. F is often described as a transition state diagram. $F : Q \times \Phi \rightarrow Q$
- q_0 and m_0 are the initial state and memory respectively.

The sequence of transitions (path) caused by the stream of input symbols is called a computation. The computation halts when all input symbols are consumed. The result of a computation is the sequence of outputs produced by this path.

3 Formal Development Framework based on X-machines

Building a formal framework using X-machines as a modelling language [9] for aiding the development of safety critical systems but also any other computer based system, offers the advantage of: (a) a verification technique to prove the validity of the model [6], i.e. prove whether desired properties are satisfied by the model of the system, and (b) a testing strategy to check the implementation against the verified X-machine model [12].

The framework suggested in [5] to be built around X-machines is depicted in figure 2. It is argued that, by applying the proposed framework, to critical systems it is possible to assure that several "safety" properties hold in the final product. The grey shaded areas are tasks in which X-machines are used as the core formal method.

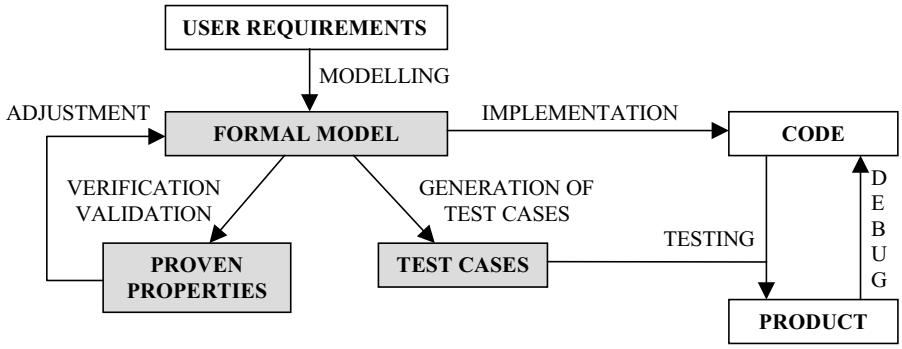

Fig. 2. A formal framework based on X-machines that supports the development of safety critical systems

First of all, from the user requirements the system is described as an X-machine model. Then a verification technique for X-machines (model checking X-machine models) verifies that certain safety properties hold in the model and feedback is used to adjust the model (or the temporal logic formulas expressing the properties). The actual implementation task produces the code written in a programming language. Finally, the testing of the implementation for correctness with respect to the model takes place and the refinement of the implementation, through the use of a complete test set derived from the X-machine model using the testing strategy that Ipate and Holcombe proposed in [11].

All the above will aid in the development of more correct and reliable computer systems, but in order to use them in industry and in real-life projects a disciplined and clear process should be devised that will enable developers to adopt the X-machine formal method. The next section describes a proposal for a new formal development methodology based on X-machines.

4 $\mathcal{XFun}$; A Lightweight Formal Development Process

The X-machine, as a tool to describe computerised systems unambiguously, provides the appropriate level of intuitiveness that fills the semantic gap between what the specifier thinks and what a formal model is. By providing a more intuitive way of modelling the functional behaviour than Z, VDM and other formal languages the difficult task of formal modelling seems as easy as possible. Thus, understanding the formal model is possible even with minimum training. Also educating people to use it as a modelling tool without requiring training in advanced mathematics is feasible. The results of analysis made from experts or educated persons are reviewable from people with no experience on the formal methods field, thus making it suitable in using it in the industry.

But only the above is not enough for the successful wide use of a formal method in the industry. Professional and international certification bodies propose or force guidelines for system development and most of them suggest the use of formal methods. For example, the UK Defence Standard 00-55 [2] supports the need for a formal method like the one described before, by requiring formalised development of software, use of animation of the requirements specification, static analysis, and dynamic analysis (meaning testing of code). Most of the work done trying to satisfy these standards is based on model based formal notations like Z [3] and VDM [1] that seemed to be difficult to communicate to people with limited knowledge of formal methods. It is also important to understand that the formalisms which provide only intuitive ways of modelling computerised systems and methods which verify the produced models are not enough. Even if the model is verified, testing is essential to prove that the mathematical model was appropriate and any assumptions made during the modelling of the system under development were correct and that the final product meets the user requirements.

The proposed formal framework built around the X-machine computational model appears to be a promising one because it has all the above mentioned characteristics. However, it is still not very clear how this will be practical and which is the efficient way for the industry to adopt all these ideas and apply them in real life projects. The benefits and practicality of lightweight development methodologies in general and more specifically of the UP appear to be an appropriate ally for this purpose. Considering all the above this section proposes a lightweight formal methodology for the development of computer based systems that adopts both the X-machine and UP, and presents it as a possible alternative that the companies involved in developing computer based systems either critical or not could use as a development methodology. The proposed formal methodology will be called $\mathcal{XFun}$ (pronounced extra fun). Briefly $\mathcal{XFun}$ will be described in the following paragraphs.

UP is a software engineering process that provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget [17]. Rational Unified

Process (RUP) is the most well known development methodology that follows UP. RUP adopts the UML notation as the core modelling method.

$\mathcal{XF}un$ is a development methodology that fully adopts the UP and proposes as the core modelling technique the X-machine, aiming for the development of computerised systems that will be reliable and correct w.r.t. user requirements. This is achieved through the use of several existing techniques and tools like the communicating X-machine method that enables modular modelling, the verification technique, the testing strategy, and XMDL [15] (X-machine description language) and its accompanying tools [14].

Fully adopting UP as mentioned before, $\mathcal{XF}un$ has two distinct aspects, a static and a dynamic as in UP. It follows the four proposed phases by UP, where each phase consists of a number of iterations. These phases are planned in the beginning where also the time plan of each iteration is scheduled following the guidelines that the UP proposes. Each iteration in $\mathcal{XF}un$ also ends with an executable product accompanied by several artifacts, and all these grow and mature over time through each iteration ending to the incremental production of the final product and artifacts. Disciplines, activities and roles are the same as in UP.

What is new in $\mathcal{XF}un$ is that it proposes a formal method as the cornerstone of the development of the system. All disciplines are using X-machines as a basis and all the produced artifacts are using X-machine theory but also any traditional good practice that can be combined with X-machines is used to enhance the methodology and make it more flexible and appropriate for adaptation from the industry.

4.1 $\mathcal{XF}un$ Best Practices

As the UP proposes several principles named best practices, this section presents the proposed by $\mathcal{XF}un$ best practices under the light of using X-machines as the core formalism in the development of a system using this methodology.

Visual modelling $\mathcal{XF}un$ facilitates visual modelling by using the X-machine for modelling the systems under development. X-machines combine the dynamic features of FSM with data structures enabling the modelling of both the static and the dynamic aspect of the system. The graphical notation used that is a transition graph is aiding for better understanding of complex systems and provides a means for exploring and comparing design alternatives at a low cost. With the addition of the internal data store (memory) and the set of functions that are labelling the transitions and manipulate the memory, X-machines facilitate modelling of the data part of the system and form a foundation for implementation.

Being a formal model, it describes the requirements precisely and unambiguously avoiding any misconceptions among the members of the development team, but also between the team and the clients. This is achieved because its intuitiveness enhances the communication of decisions among the members of the

development group and facilitates the invaluable communication of this group with the users.

Finally, the X-machine proved to be suitable to model systems from a lot of different domains [5], [7], [16] and its generality implies that is capable of modelling any computer based system and moreover creating elegant and intuitive models.

Component-based Architecture In order to achieve flexibility over changes and development through progressively improved and refined products following the iterations, a modular component-based development process was needed that will enable also reuse of available off the shelf components. The modular communicating X-machine system presented in [13] provides a modelling tool based on the X-machine, where a complex system can be decomposed in smaller components that can be modelled as simple X-machine models and at the end the communication of all these components can be specified to model the complete system as a communicating X-machine model.

With this method it becomes possible to reuse off-the-shelf X-machine models but also to model only once a component and use many of its instances in the final communicating system as instances of the X-machine component model decreasing the development time and increasing reliability. All the component X-machines are verified and tested, thus increasing confidence in using them aiming towards a correct final product.

Requirements Management $\mathcal{XF}un$ employs the modular communicating X-machines approach in order to manage requirements in a systematic and continuous manner, aiming to elicit and document the requirements of the system and establish and also maintain an agreement between the customer and the development team on the system's changing requirements.

In this approach, the X-machine component models are the basis for the entire development process and the final model of the system under development is a communicating X-machine. The memory structure together with the functions offer a flexibility to the X-machine model regarding changes in the requirements. This characteristic makes the X-machine a powerful modelling tool in $\mathcal{XF}un$. It is the cornerstone that satisfies the requirement of the process to use a modelling method that is flexible in changes of the user requirements. By changing the set of inputs, the memory and maybe also the functions, it is possible to alter the model in order to include any changes of the requirements. The designer could use the memory element of the X-machine whenever there is a prediction that a user requirement might change and this way only alterations in the functions should take place to model the changes. This in addition to the flexibility offered by the modular development facilitated by the communicating X-machine system allows gradual development and flexibility in changes.

But it is not only important the modelling tool to be flexible in changes but also aid the developer to elicit the requirements from the users. With the use of tools that are built around the XMDL language it is possible if the X-machine

model is written in XMDL to syntactically check it and then automatically animate this model [14]. Through this simulation it is possible first of all for the developers to informally verify that the model simulates the actual system under development, and then also to demonstrate the model to the users aiding them to identify any misconceptions regarding the user requirements between them and the development team. Also existing and well established techniques in software engineering, like prototyping, can be used as complementary methods to assist in this phase.

Change Management It is clear that developing large scale computer systems requires disciplined control over the management of development teams, iterations, releases, artifacts etc. Lack of such control will lead to unpleasant situations ending with the failure of the whole project. The UP in combination with the communicating X-machines practical and modular approach provides the required disciplined control in the methodology.

Project and risk management are facilitated and supported in $\mathcal{XF}un$. It could adopt, mainly in the inception phase, mature and standard hazard and risk analysis, qualitative risk assessment and assurance techniques. This fulfills the need for integration of formal software development techniques with such hazard, risk and project management techniques [18] in order to build a complete development methodology of critical also computer based systems.

The modular approach proposed using the communicating X-machines enables the project manager to decompose efficiently the whole system and assign the development of components in different teams. This modular approach supports an iterative gradual development and facilitates the reusability of existing components, making the management of the whole project more flexible and efficient, achieving its completion with lower cost and less development time. Of course the benefit is not significant when the decomposition of the model into communicating X-machines it does not match the functional decomposition of the system, something that the flexibility of the X-machine is aiding the developers to avoid.

Quality Verification The fact that software problems are much more expensive to find and fix after deployment than before, forces the modern development methodologies to provide mechanisms that ensure the correct development of the systems throughout the life cycle that also gradually minimise the risk from the early stages of development. Henceforth, quality should be continuously assessed. $\mathcal{XF}un$ provides an intuitive modelling method that with the use of tools aids in the elicitation of the user requirements. But in order to increase the confidence that the proposed model has the characteristics that the user wants, an automatic and formal verification technique is provided. This formal verification technique for X-machine models enables the designer to verify the developed model against temporal logic (an extended version was devised appropriate for X-machine models, named $\mathcal{X}m$CTL that is presented in [6]) formulas that express the desired by the user and by the designer properties that the

system should have. Using this technique the designer is confident that the produced model has the desired properties and once the properties are expressed in $\mathcal{X}m$CTL formulas, whenever a new version of the model exists the new model is verified using this automated technique.

But that is not all. $\mathcal{XF}un$ supports not only static but also dynamic analysis. Early in the development the first versions of the system are implemented as the methodology commands. By having the X-machine model, from the very first version it is possible to use the testing strategy to test the implementation and prove its correctness with respect to the X-machine model. The model's correctness with respect to the user requirements has been demonstrated with the use of the formal verification technique. This way by the end of each iteration the developers are confident that their product is correct with respect to the user requirements.

Therefore, with the continuous verification and testing from the early stages risks are reduced and the developer is confident for the correctness of the system under development throughout the whole process. It worths noticing that components that have been verified and tested can be reused without any other quality check and the proposed communicating X-machine system supported by the methodology is based in the idea of reusability, thus minimising the development time without risking the quality of the product.

Iterative Development As stated before the $\mathcal{XF}un$ methodology starts with a planning phase where the several iterations that will follow until the end of the project are scheduled. In the first iterations focus is given in the user requirements, project and risk management and hazard analysis. Every iteration includes the following activities:

- **User requirements.** The activity where the developers understand the properties of the system under development, using mainly traditional techniques from software engineering, like interviews with the users, questionnaires etc. depending on the project, but also using the animation tool to animate early versions of the X-machine model for enhancing the clients - developers communication.
- **Modelling.** In this one the X-machine model is created. The system is decomposed in the initial phases and gradually the model is finalised in later stages. Focus is given to this discipline mainly in the first two phases (inception and elaboration).
- **Verification.** The X-machine model is verified by proving that the desired properties expressed as $\mathcal{X}m$CTL formulas are satisfied by the model, using the model checking for X-machines technique.
- **Implementation.** The actual development of the software considering as design the X-machine model with the possible addition of other semi-formal models (like UML diagrams) that will make the design more complete and detailed.
- **Testing.** The implementation is checked with respect to the X-machine model using the formal testing strategy, and finally

– **Evaluation.** The release is evaluated.

Each iteration ends with a version of the system under development. At the end of all iterations the last phase of the project follows which is the deployment. The whole development is based on the iteration of a cycle composed of the same activities which at the end always produces a version of the final product.

4.2 $\mathcal{XF}un$ advantages

The proposed formal development methodology, namely $\mathcal{XF}un$ has several valuable advantages that are a combination of the advantages the UP and the X-machine exhibits:

– $\mathcal{XF}un$ is adaptable with respect to the system under development. Thus it can play the role of a development methodology of any computer based system and more importantly it facilitates the development of correct critical systems. This belief is based first of all on the fact that in the $\mathcal{XF}un$ methodology there is room available for risk and hazard analysis whenever this is vital for the development of the system. Secondly, it is important to distinguish the qualitative difference between the most important safety requirement, that is the absence of accident, and the reliability requirement, which is the continuity of the required function. So, although usually overlapping, system safety and software reliability could be distinct or in some cases conflicting. Reliable software cannot assure that the system that is using it is safe [18]. Safety is a system property, so modelling the whole system and validating it is very important and not isolating the software from the system. $\mathcal{XF}un$ by employing the communicating X-machine as a modelling method, gives the ability to model the whole system and verify the desired properties for this model. On the parts that are software a formal testing strategy is followed also to improve the reliability of the software.
– The communicating X-machine method supports a disciplined modular development, allowing the developers to decompose the system under development. Decomposition aids in handling large scale systems, thus $\mathcal{XF}un$ fulfils one basic requirement in order to be used as a development methodology in the industry.
– Also communicating X-machines provide the appropriate style of developing models reusing off-the-shelf existing verified component models and maybe even their tested implementations. This way development time is minimised, allowing together with the appropriate decomposition spreading the workload so that developers are able to deliver a product of high quality on time through continuous integration.
– Confidence in the correctness of the product is built in any iteration through the use of verification and testing techniques and the product is progressively improved and refined providing evidence of projects status to customers.
– This way risks reduce from the early phases of the project opposite to the heavy methodologies where risk reduces at the end of the project when it might be too late and fatal for the success of the project.

– One of the major advantages that formal modelling provides is that descriptions are unambiguous. The developers understand the system better as a result of trying to describe it unambiguously. Also, an intuitive formal method, like the X-machine, makes possible for the simple user to understand the documents produced by the developers. This enhances the communication between the user and the developers allowing feedback from the users.
– Tools, like automatic animation of the model, help the user to monitor the model proposed by the developers allowing more complete and immediate feedback. As a consequence at the end of each iteration the user provides valuable feedback early in the development process.
– The development team and the users learn while developing, and all learn fast due to the intuitiveness of the X-machine formalism. This improves the whole process.
– Changing requirements is an expected event through the development time and can be handled efficiently using the modularity of the communicating X-machine model and the flexibility of the X-machine component model.

5 Conclusion

The demand on the one hand for more complex but also more reliable and correct computer based systems, and on the other hand the fact of several changes of the user requirements through the development cycle, leads to the need for more formal but also agile development methodologies. $\mathcal{XF}un$ is a new proposal for a lightweight development methodology of computer based systems based on formal methods, with very promising advantages, that will improve the final product as being more reliable, safe, robust and correct w.r.t. user requirements. It is flexible and "armed" with a lot of formal techniques that will ensure the development of more correct systems. But in order to be most effective the system under development should be a component based reactive system.

Also in order to be actually adopted by industry all the formal techniques that support $\mathcal{XF}un$ should mature, be finalised stable versions, be supported by complete, user friendly tools that are compatible with the existing tools used by the industry and prove their abilities through the application on many real case studies. Thus $\mathcal{XF}un$ is for now a proposal that forms the base for a complete and mature methodology to appear in the future. But it is very important that the initial framework has been established, because $\mathcal{XF}un$ during its evaluation through experimental projects will aid the existing and the newly proposed techniques build around X-machines to mature. Then as these techniques mature and integrated tools will be built for them, they will aid $\mathcal{XF}un$ to become an industrial strength methodology.

References

1. *Information technology - Programming languages, their environments and system software interfaces - Vienna Development Method - Specification language - Part 1:*

Base language, volume ISO/IEC 13817-1. International Standards Organization, 1996.
2. *Requirements For The Procurement Of Safety Critical Software In Defence Equipment*, volume 00-55/Issue 2. Ministry of Defence, UK, 1997.
3. *Z Notation, Final Committee Draft*, volume CD 13568.2. International Standards Organization, 1999.
4. S. Eilenberg. *Automata Machines and Languages*, volume A. Academic Press, 1974.
5. G. Eleftherakis. A Formal Framework for Modelling and Validating Medical Systems. In V. Patel, R. Rogers, and R. Haux, editors, *MEDINFO 2001*, volume 1, pages 13–17, London, UK, September 2001. IOS Press.
6. G. Eleftherakis, P. Kefalas, and A. Sotiriadou. XmCTL: Extending temporal logic to facilitate formal verification of X-machine models. *Analele Universitatii Bucuresti, Matematica-Informatica*, 50:79–95, 2001.
7. G. Eleftherakis, A. Sotiriadou, and P. Kefalas. Formal Modelling and Verification of Reactive Agents for Intelligent Control. In *12th Intelligent Systems Application to Power Systems Conference (ISAP03)*, Lemnos, Greece, September 2003. IEEE Power Engineering Society.
8. M. Fowler. The new methodology, thoughtworks. http://www.martinfowler.com/articles/newMethodology.html, April 2003.
9. M. Holcombe. X-machines as a basis for dynamic system specification. *Software Engineering Journal*, 3(2):69–76, 1988.
10. M. Holcombe, K. Bogdanov, and M. Gheorghe. Functional test generation for Extreme Programming. In *Second International Conference on eXtreme Programming and Flexible Processes in Software Engineering (XP2001)*, pages 109–113, 2001.
11. M. Holcombe and F. Ipate. An integration testing method that is proved to find all faults. *International Journal of Computer Mathematics*, 63(3):159–178, 1997.
12. M. Holcombe and F. Ipate. *Correct Systems: Building a Business Process Solution*. Springer Verlag, London, 1998.
13. 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.
14. P. Kefalas, G. Eleftherakis, and A. Sotiriadou. Developing Tools for Formal Methods. In *9th Panhellenic Conference on Informatics*, Thessaloniki, November 2003.
15. P. Kefalas and E. Kapeti. A design language and tool for X-machines specification. In D.I. Fotiadis and S.D. Nikolopoulos, editors, *Advances in Informatics*, pages 134–145. World Scientific Publishing Company, 2000.
16. E. Kehris, G. Eleftherakis, and P. Kefalas. Using X-machines to model and test discrete event simulation programs. In N. Mastorakis, editor, *Systems and Control: Theory and Applications*, pages 163–168. World Scientific and Engineering Society Press, July 2000.
17. P. Kruchten. *The Rational Unified Process, An Introduction*. Addison Wesley, Reading, MA, 2 edition, 2000.
18. N.G. Leveson. *Safeware: System Safety and Computers*. Addison Wesley Longman, 1995.