THE USE OF THE ASPECT ORIENTED PROGRAMMING (AOP) PARADIGM IN DISCRETE EVENT SIMULATION DOMAIN: OVERVIEW AND PERSPECTIVES

Meriem Chibani(1)- Brahim Belattar (2) and Abdelhabib Bourouis(3)
(1) (3) Department of Mathematics and Computer Science
University of Oum El Bouaghi, Oum El Bouaghi, 4000, Algeria
(2) Department of Computer Science, University of Batna, Batna, 5000, Algeria
chibani_meriem@live.fr brahim.belattar@univ-batna.dz a.bourouis@univ-oeb.dz

ABSTRACT

The problem of realizing the Separation Of Concerns (SOC) principle is at the heart of the software crisis where crosscutting concerns tend to produce a messed simulation systems by decreasing their modularity, and reducing their reuse. Aspect-Oriented Programming paradigm is newly technique introduced for software development. It may co-exist with other methodologies, such as object-oriented, in order to keep the most system interesting properties, such as reusability and adaptability. In this paper, we present a synthetic study of the most efforts of using the aspect oriented programming paradigm (AOP) in the simulation modeling field. Besides that, we explore the promising perspectives in this research area and our ongoing work towards an Aspect Oriented Simulation Framework (Japrosim).

KEYWORDS

Separation of concerns, aspect oriented programming, simulation modeling, simulation frameworks, Japrosim.

1 INTRODUCTION

Modern applications including simulation modeling software require more and more features in order to cope growing problems. They could include many concerns for instance: synchronization (many simulation environments implements concurrency, trying to provide a more direct way of modeling real systems), scheduling (for example, in discrete-event based simulations), optimizations (because performance is a desired property in simulation), distribution (for example, using web-based simulation) and so forth [1]. The presence of these various concerns penalizes evolving capacities of the application. Indeed, their implementation in the object paradigm is often crossing the class hierarchy and tends to produce a messed design [2].

The distinction between the different categories of concerns, will simplify design and implementation, give better understanding, decrease coupling between concerns (strong cohesion) and more generally, increase reuse.

It is important that the separation of concerns (SOC) is achieved both at the conceptual level and the implementation level. The object oriented paradigm does not satisfy these assumptions, whereas it provides a powerful way to separate core concerns, it will leave something to be desired when it comes to crosscutting concerns. To overcome this shortcoming, several methodologies as generative programming, meta-programming, reflective programming, compositional filtering, adaptive programming, subject-oriented programming, aspect-oriented programming and intentional programming have emerged as possible approaches to modularizing cross-cutting concerns [3]. AOP is the most popular among these methodologies and many of work that lead to it today is done at universities all over the world.

Cristina Lopes and Gregor Kiczales of the Palo Alto Research Center (PARC), a subsidiary of Xerox Corporation, are among the early contributors to AOP. Gregor coined the term “AOP“ in 1996. He led the team at Xerox that created AspectJ, one of the first practical implementations of AOP, in the late 1990s. Xerox recently transferred the AspectJ project to the open source community at eclipse.org, which will continue to improve and support the project.

AspectJ is based on Java, but there are implementations of AOP for other languages, like AspectC for C and Pythius for Python, that apply the same concepts that are in AspectJ to other languages. Furthermore, there are others Java implementations of AOP than AspectJ, such as Java Aspect Component (JAC), Caesar, Jiazzi, JBossAOP, AspectWerkz and JasCo. These implementations differ in the ways of expressing the crosscutting concerns and in weaving mechanisms to form the final system [4].

The rest of the paper is organized as follows. Section 2 describes briefly the aspect oriented programming paradigm. Section 3 discusses the state of the art. Section 4 presents the most active research areas and points some directions to our future work and finally a conclusion is given in Section 5.

2 THE ASPECT ORIENTED PROGRAMMING (AOP)

The aspect-oriented programming paradigm is a newly introduced methodology for software development. It is not supposed to replace the wide-spread Object-Oriented programming methodology but extends it. The AOP could be seen as a complementary technique that co-exists with other approaches, such as object-oriented, procedural, functional or event-driven programming [5]. Like any technology, AOP goes through the Gartner Hype Cycle which offers a more accurate gauge of the benefits and risks that the technology is likely to offer for her users. The AOP's Hype cycle, as shown in Figure 1, is divided into five major phases: technology trigger, peak of inflated expectations, trough of disillusionment, slope of enlightenment and plateau of productivity. A detailed discussion of the cycle could be found in [6]. Currently, AOP is on the plateau of productivity phase where it is applied progressively in the real-world application.

Like any other programming methodology, an AOP implementation consists of two parts:

- The language specification: used for programming basic and non-functional concerns of application and describing rules weaving which specify how to integrate the implemented concerns in order to form the final system. The specificity of the weaving rules determines the amount of coupling between the aspect and core logic once the weaving rules have been applied. A standard language like java, c++ and c is suitable for programming both parts. However, the use of other languages, like XML, for weaving rules is also acceptable.
- The language implementation: verifies the code's adherence to the language specification and translates the code into an executable form according to the weaving rules. This is commonly accomplished by a compiler or an interpreter like AspectJ weaver [7].

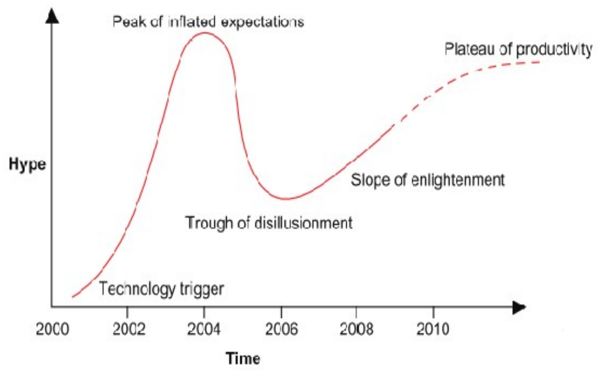

Figure1. The AOP's Hype cycle [6].

To implement crosscutting concerns, an AOP system may include many concepts where the central one is the join point model which consists of two parts: join points, the points in the execution of an application; and pointcuts, a mechanism for selecting join points. The aspects have methods (advices) that are attached to one or more join point. When a method is attached to join points, it will be executed, beside that, it has a modifier which may specify the execution time relative to the join points: before, after, around, after exception or also after return value. Moreover, these methods have an additional instance variable named thisJoinPoint that encapsulates the contextual information captured from the current junction. Advice is a form of dynamic crosscutting because it affects the execution of the system. Furthermore, AOP implementation may contain static crosscuttings which alter static structure of the system. For example, when implementing tracing, you may need to introduce the logger field into each traced class; inter-type declaration constructs make such modifications possible. In some situations, you may need to detect certain conditions, typically the existence of particular join points, before the execution of the system; weave-time declaration constructs allow such possibilities. Figure 3 shows all these concepts and their relationships to each other in an AOP system. Each AOP system may implement a subset of the model. For example, Spring AOP doesn't implement weave-time [6].

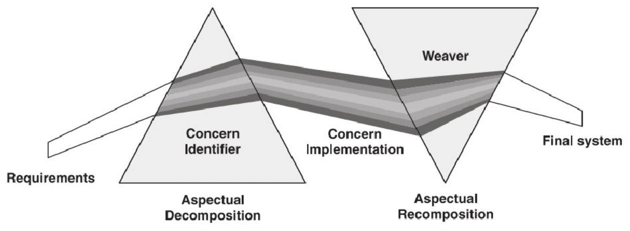

Figure2. AOP development stages [4]

In many ways, the process of developing a system using AOP contains three phases: identification of concerns, their implementation and development of the final system by combining them in the following way:

- Aspectual decomposition: decomposes the core-level from crosscutting concerns. This phase is compared with the passage of a beam of light through a prism to separate its different color components.
- Concern implementation: implements each concern independently by using procedural or OOP techniques for the core concern.
- Aspectual recomposition: specifies the recomposition rules by creating modularization units or aspects using an aspect integrator or weaver. By analogy and as shown in Figure 2, we could compare this step to a new passage of color components in a prism that combines them to make out a single ray of light [4].

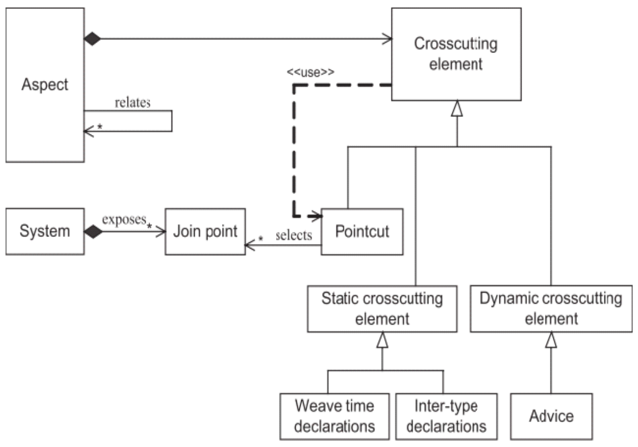

Figure3. Generic model of an AOP system [6].

3 STATE OF THE ART

A considerable number of simulation modeling software applications based on aspect oriented programming paradigm (AOP) have been proposed, among them we mention:

The multi-agent simulation system discussed in [5] and [8], which consists of two types of agents: agents for the description of the simulation model and agents for observational mechanisms. This collection of independent agents is interacting with discrete events where every agent has a schedule that generates its plan of activities. The system is executed on Telemodeling framework which uses an object-oriented paradigm to define its muli-agent models and web technology to make ease to use modeling and simulation environment; the kernel of this framework is the programming language MAML[1] (Multi-Agent Modeling

1 MAML is a domain specific programming language that is designed to build computer simulation based on Swarm simulation package

Language) which has the capacity of the dissociation between the model and the observational mechanisms, thanks to the aspect oriented paradigm, from the design phases until the implementation phase thereby increasing the maintainability of the system and decrease its complexity. MAML has the xmc compiler which generates Objective-C code from the MAML source code after weaving the model object and observation object. Although the richness of MAML syntax to support AOP is remaining in its infancy if it is compared with the AspectJ language.

In [9], the authors use the aspect oriented paradigm to develop a multi-agent system dedicated to simulate physical phenomena. The MAFES system (Multi Agent Finite Environment System) consists of an environment in the form of the node matrix and a set of agents operating on these nodes where aspects are used to assign tasks to agents by adding appropriate functionality to perform their task. In addition, the aspects weave the appropriate resources and attributes to the environments nodes. MAFES contains three others type of aspects for control, visualization and storing simulation results. MAFES is implemented using the AspectJ language. The implementation of the Aspect Oriented Programming makes the system generic to build versions of the system for specific requirements (it is enough to weave appropriate aspects). The authors experiment their system with two phenomena. The first is heat exchange and motion phenomena. The second is heat exchange and crystallization.

In [10], Daesung and Kang propose a simulation based design level performance analysis method where the performance concern of the software application is separated from the functionality model since the design phase by the use of aspect oriented programming. Unlike the design of the software system which is modeled using UML class diagrams and sequence diagrams, the performance model is an XML-based representation derived from the UML performance profile. After code generation from the design model, the AspectJ weaver is introduced in order to formulate the simulation code. The authors experiment their approach using a distributed map viewer system. In [11] they argue that their approach is generic and could be used for analysis of other quality attributes of software systems as reliability and performance concerns. This work seems satisfactory, but it does not take into account the architecture on which the application is running. In [12], another approach for analyzing performance is proposed. Unlike the precedent approaches, it defines performance as a collection of aspects which include response time, rate throughput, probability, time between errors and so on metrics over aspect-oriented formal design analysis framework (FDAF). The authors focus on modeling the response time performance aspect and they have chosen real-time UML as the base notation which is translated into Architecture Description Language (ADL) Rapide. The authors use the simulation technique to evaluate response time for the DNS query processing subsystem by using Rapide's analyzing tools. In [13], Hui and Dorina propose an approach for performance analysis of UML models based on Aspect-Oriented Modeling (AOM) techniques. In AOM, an aspect-oriented design model consists of a set of aspects and primary models. An aspect model describes how a single objective is addressed in the design, while the primary model addresses the core functionality of the system as given by the functional requirements. Weaving rules are used to weave aspect models with the primary model. These rules are stored separately from the aspect and the primary model, which makes both the aspect models and the rules reusable. The aspects and the primary model are composed before implementation or code generation [14]. Hui and Dorina approach studies the performance effects of concrete aspects (e.g. security features) on the overall system performance (in this paper the application is named the Document Exchange Server (DES)). The approach proceeds by adding performance annotations to both the primary and aspect models using the UML performance profile then instantiating the generic aspect model thereby converting the aspect security model into a context-specific one, by following a set of binding rules provided by the designer which transforms the parametric annotations of the generic aspect model into concrete ones. The latter is composed with the primary model, according to a set of composition directives. The result is a composed annotated UML, which can be transformed automatically into a performance model (Layered Queueing Networks (LQN) in this case) by using the transformation techniques. The LQN model is analyzed with an existing solver.

In [15], the authors discuss a new approach for separation of functional (qualitative) behavior and quantitative performance constraints since the specification phase. Thanks to AOP, the aspects of a specification are written in different languages, the process algebra LOTOS for an abstract specification of functional behavior and the probabilistic temporal logic for quantitative aspects (performance constraints). The aspect weaving composes the two aspect specifications and the result of this composition is an automata-style global model which can be generated from the composition of a labeled transition system (derived from LOTOS). Event schedulers are derived from temporal logic formulae. Finally the global model can be used for performance analysis based simulation.

In [16], the authors address a new aspect-oriented approach for disaster prevention simulation system (ABR[2]). The proposed approach separates the core functionality of the simulation application from simulation cross-cutting concerns thanks to Horizontal decomposition (HD) method which relies on the aspect-oriented programming paradigm. The approach is implemented on AoSiF (Aspect-oriented Simulation Framework) which is an extension of distributed simulation framework (DiSiF) [17] where it use the resource paradigm, actor based workflow modeling, web services and Grid computing as implementation technology and Java Annotations for declarative programming in addition to AspectJ for the aspect-oriented implementation. To demonstrate the applicability of their approach, they implemented two cross-cutting concerns, namely distribution and tool integration. In [18] the same authors with Alexandru complete the implementation of other cross-cutting concerns: Job Execution and Fault-Tolerance in addition to tool integration concern. The approach affirms its efficiency in the term of increasing the reusability and the maintainability through experiments in the DAS-3[3] multi-cluster grid.

Judicaël and Olivier [19] discuss the development of advanced simulation scenarios by means of new software engineering techniques: Aspect Oriented Programming paradigm and the Architecture Description Language (ADL) designed for the Fractal component model. These new techniques enforce the separation between models and scenarios which allows better reuse of components in both parts: reuse of a given model with various scenarios, or reuse of a given scenario with various models in order to save time, money and human effort. The authors use the Open Simulation Architecture (OSA) which is a discrete-event simulator, to illustrate the benefits of the previous techniques by simple use cases of network security studies.

In [20], an industrial strength Model-Driven Engineering engine, the Motorola WEAVR including the Telelogic TAU modeling and simulation environment is presented. WEAVR combines Aspect-Oriented Modeling and Automated Code Generation technologies. Where fully weave aspects passes at the modeling level. WEAVR uses translation-oriented style of UML 2.0 modeling and including a novel joinpoint model for transition oriented state machines. This engine permits weaving at the modeling level which is advantageous over code level weaving, because aspects can be woven according to different joinpoint models. WEAVR benefits from MDE and Aspect-Oriented Software Development (AOSD) which exhibit some complementary properties. Modeling enables systems to be specified at higher level of abstraction but suffers from difficulties with respect to the refinement and integration of system perspectives. On the other hand, aspect technologies focus on the modularization and composition of concerns, but lack appropriate abstraction mechanisms [21].

In [22], the authors implement the conduit simulator which uses the AOP paradigm based on

2 ABR serves as nearly warning system for emergency situations at nuclear power plants. It calculates the concentration of pollutants substances in the atmosphere

3 DAS-3, The Distributed ASCI Supercomputer 3, is a wide-area distributed cluster designed by the Advanced School for Computing and Imaging (ASCI) in The Netherlands http://www.cs.vu.nl/das3.

Logic Metaprogramming (LMP) at the code level. The simulator implements its cross-cutting concerns (synchronizing and order of execution, user interface (UI) and logging) in different aspect-specific languages (ASLs) and thanks to logic modules which encapsulate aspects and the implementation of ASLs. The simulator gains a modular aspect-weaver mechanism that offers the generality of a general-purpose aspect language without losing the ability and advantages of defining aspects in aspect-specific languages. The simulator uses the SOUL/Aop system which is a prototype aspect-weaver in Smalltalk.

In [23], the aspect oriented paradigm is used at scenario-based requirements level where interactions model with aspects in a way that they can be immediately validated using simulation methods. Scenarios are modeled as UML sequence diagrams (non-aspectual interactions) and interactions that cross-cut other interactions are modeled as Interaction Pattern Specifications (IPSs) which describe a pattern of structure or behavior and they are defined in terms of roles. After the aspectual and non-aspectual interactions are composed by instantiating the aspects the resulting set of interactions translate automatically into a set of executable UML state machines. Whereas in [24] the composition is done at the state machine level. The authors used Error-handling crosscutting concern to illustrate their technique.

In [25] new verification method based on simulation is discussed. The method aims to automate the coverage analysis[4] using Aspect-Oriented programming. Aspects here are de-scribed as checkers a crossing the system for logging the state transition information over the time during the simulation. In this approach, the Unified Modeling Language (UML2) is used to describe SystemC models and AspectC++ language for aspects implementation.

4 FUTURE ISSUES

As can be seen from the above discussion, there are many new approaches proposed that support the AOP according to several metrics:

1) Application level:
   - Specification phase.
   - Design phase.
   - Implementation level.
2) The used technology:
   - Logic Metaprogramming (LMP).
   - Model Driven Engineering (MDE).
   - Aspect-Oriented Modeling (AOM) techniques which current research is addressing the following problems: using aspects to describe crosscutting concern solutions; describing aspect models at different levels of abstractions (e.g. generic and mechanism specific); composition of aspect and primary models; automation of the AOM approach; analysis of composed models to identify and resolve conflicts and undesirable properties that may arise as a result of the composition [13].
3) Aspect level weaving :
   - Weaving at the modeling level.
   - Code level weaving.
4) The host methodology:
   - The AOP co-exist with Object-Oriented methodology.
   - Multi-agent system (MAS).
   - Programming with Components.
5) The goal of using AOP in simulation:
   - Development of Complex Simulation Software.
   - Separation between model and scenarios.
   - Separation between model and observational mechanisms (visualization of simulation result and user interface).
   - Development of generic framework.

In any way however the approach used to apply AOP in the discrete event simulation domain the main goal is the design of new systems based on the less is more principle in the sense that now a method of complete separation of the cross-cutting concerns from the core functionality is sought. This way the focus is shifted toward the

4 The coverage analysis is a measure used in software testing. It describes the degree to which the source code of a program has been tested.

development and maintenance of the application core rather than of cross-cutting concerns. The core functionality represents less code but more added value since it is the actual product of the research institute [16].

We could consider the discrete event simulation as an interesting field for the exploitation of AOP's advantages and its recent research themes which are illustrated by the diagram in Figure 4.

In order to apply the separation of concerns (SOC) principle in the discrete event simulation domain and to decrease the complexity of simulation modeling software applications, increase the maintainability, adaptability, reliability and modularity; we aim to develop new methodology for discrete event simulation based on aspect oriented paradigm (AOP) and applying it for the existing open source JAPROSIM [26]. This framework is a general purpose Discrete Event Simulation library, implemented in Java specifically for Process Interaction world view. It is under active development and several new modules are conceived and to be implemented like graphical animation, steady state detection, graphical modeling, explanation, and advanced results analysis.

There exist many of AOP implementations for several modern programming languages but AspectJ, the de facto standard AOP language which is implemented on top of Java, seems to be the most mature and is the best suited for our future work.

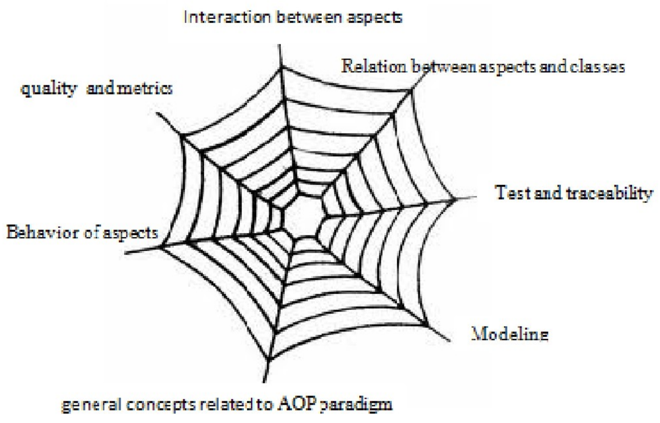
Interaction between aspects
Relation between aspects and classes
quality and metrics
Test and traceability
Behavior of aspects
Modeling
general concepts related to AOP paradigm

Figure4. AOP research areas [27].

5 CONCLUSION

AOP is relatively new paradigm in computer programming that provides many advantages among them:

- Increasing productivity, the use of aspects avoids having any rewrite (redundancy) and redesign parts of a program.
- Reducing programs complexity by separating the code in various sections (modularity), the complexity is reduced.
- Improving software quality at different points of view (maintenance, test, and reuse).

The implementation of the aspect oriented programming paradigm in the simulation modeling field is considered as a promising research area where several approaches have already been proposed according to the following points: AOP's applying level, aspect's weaving mechanisms, the technology used, the host methodology and AOP's using goal. We aim to develop our methodology for discrete event simulation environment that exploit the richness of the AOP paradigm.

6 REFERENCES

1. Díaz, J.A. Pace, Campo, M.R., Fayad, M.E.: A Language for Simulation: Bringing Separation of Concerns to the Front. Aspects and Dimensions of Concerns Workshop, 14th ECOOP '00, Sophia Antipolis and Cannes, French (2000).
2. Kiselev, I.: Aspect-Oriented Programming with AspectJ. Manning Publications, (2003).
3. Kiczales, G., Lamping,J., Mendhekar, A., Maeda ,C., Lopes, C. V., Loingtier, J.M., Irwin, J.: Aspect-Oriented Programming. In proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland. LNCS, vol. 1241, pp. 220--242. Springer-Verlag (1997).
4. Laddad, R.: AspectJ in Action Practical Aspect-Oriented Programming. Manning Publications, (2003).
5. Gulyás, L., Kozsik, T.: Aspect-Oriented Programming in Scientific Simulations, Proceedings of The Sixth Fenno- Ugric Symposium on Software Technology, Estonia (1999).
6. Laddad, R.: Aspectj in Action: Enterprise AOP with Spring Applications. Manning Publications, Second Edition (2009)
7. Choudary, K.: Introduction to Aspect Oriented Programming, School of Info. Tech., (2005).
8. Gulyás, L., Kozsik, T., Corliss, J.B: The Multi-Agent

Modelling Language and the Model Design Interface, Journal of Artificial Societies and Social Simulation vol. 2, no. 3, (1999)

9. Bieniasz, S., Ciszewski S., Śnieżyński B.: Multi-agent Simulation of Physical Phenomena by Means of Aspect Programming, Computational Science – ICCS 2006: 6th International Conference reading : UK, May 28–31, 2006 : proceedings. Pt. 3 / eds. Vassil N. Alexandrov — Berlin; Heidelberg: Springer-Verlag, 2006. — (Lecture Notes in Computer Science; LNCS 3993). — S. 759–766. — Bibliogr. s. 765–766, Abstr. (2006).
10. Park, D., Kang, S.: Design Phase Analysis of Software Performance Using Aspect-Oriented Programming, 5th International Workshop on Aspect-Oriented Modeling, Lisbon, Portugal. (2004)
11. Park, D., Kang, S., Lee, J.: Design Phase Analysis of Software Qualities Using Aspect-Oriented Programming, In Yeong-Tae Song, Chao Lu, Roger Lee, editors, Seventh International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD 2006), 19-20 June 2006, Las Vegas, Nevada, USA. pp 29--34, IEEE Computer Society, (2006).
12. Cooper, K., Dai, L., Deng,Y.: Modeling Performance as an Aspect: a UML Based. Approach, in The 4th AOSD Modeling With UML Workshop (2003).
13. Shen, H., C. Petriu, D.: Performance Analysis of UML Models Using Aspect-Oriented Modeling Techniques, Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems MoDELS'05, (2005).
14. Lengyel, L., Levendovszky, T., Charaf, H.: Aspect-Oriented Techniques in Metamodel Based Model Transformation, International Symposium of Hungarian Researchers on Computational Intelligence, Budapest, Hungary, (2005).
15. Blair, L., Blair, G., Andersen, A.: Separating Functional Behaviour and Performance Constraints: Aspect Oriented Specification, techreport, (1998).
16. Ionescu, T.B., Piater, A., Scheuermann, W., Laurien, E.: An Aspect-Oriented Approach for the Development of Complex Simulation Software, Vol. 9, No. 1, (2010).
17. Piater A., Ionescu, T.B., Scheuermann, W.: A Distributed Simulation Framework for Mission Critical Systems in Nuclear Engineering and Radiological Protection, International Journal of Computers Communications & Control, ISSN 1841-9836, Volume:3, Supplement: Suppl.S pp:448-453, (2008).
18. Ionescu, T.B., Piater, A., Scheuermann, W., Laurien, E., Iosup, A.: An Aspect-Oriented Approach for Disaster Prevention Simulation Workflows on Supercomputers. Proceedings of the 2009 13th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications, pp 21--30, IEEE Computer Society (2009).
19. Ribault, J., Dalle, O.: Enabling advanced simulation scenarios with new software engineering techniques. In 20th European Modeling and Simulation Symposium (EMSS 2008), Briatico, Italy, pages 6p, (2008).
20. Cottenier, T.: Aspect-Oriented Modeling and simulation, tutorial is available at: http://www.iit.edu/concur/weavr/documentation.html.
21. Zhang, J., Cottenier, T., Berg, A.V.D, Gray, J.: Aspect Composition in the Motorola Aspect-Oriented Modeling Weaver, in Journal of Object Technology, vol. 6, no. 7,Special Issue. Aspect-Oreinted Modeling, August 2007, pp 89-108. (2007).
22. Brichau, J., Mens, K., Volder, K. D.: Building Composable Aspect-specific Languages with Logic Metaprogramming. Proceedings of the 1st ACM SIGPLAN/SIGSOFT conference on Generative Programming and Component Engineering GPCE'02, Vol. 2487, pp. 110-127. (2002).
23. Araújo, J., Whittle, J., Kim, D.K.: Modeling and Composing Scenario-Based Requirements with Aspects. Proceedings of the 12th IEEE International Conference on Requirements Engineering RE '04, pp 58--97. IEEE Computer Society Washington, DC, USA (2004).
24. Whittle, J., Araújo, J., Kim, D.K.: Modeling and Validating Interaction Aspects in UML. 4th AOSD Modeling With UML Workshop, na 6th International Conference on the Unified Modeling Language (UML2003), San Francisco, USA, ACM (2003).
25. Chen, Y., Qiu, W., Zhou, B., Peng, C.: An Automatic Test Coverage Analysis for SystemC Description Using Aspect-Oriented Programming, The 8th International Conference on Computer Supported Cooperative Work in Design Proceedings, (2004).
26. Bourouis, A., Belattar, B.:"JAPROSIM: A Java Framework for Discrete Event Simulation", in Journal of Object Technology, vol. 7, no. 1, January-February (2008).
27. Tessier, F.: quality assurance and development of aspect oriented software development: a model based detection of conflicts between aspects, Quebec University (2005).