

Framework for Modeling Performance in Multi Agent Systems (MAS) using Aspect Oriented Programming (AOP)

Tariq Mehmood, Naveed Ashraf,
{tmmalik, meetnaveed}@gmail.com

Department of Computer Science, International Islamic University,
H-10 Islamabad, Pakistan.

Khalid Rasheed , S. Tauseef-ur-Rehman
{drkhalid., touseef}@iiu.edu.pk

Abstract

Multi-agent systems (MASs) encompass multiple features, which tend to be scattered in the software Artifacts produced through the system modeling. As the agent complexity increases functional and nonfunctional concerns cannot be modularized based only on OO abstractions. These concerns tends to be spread across several system components, classes and methods, which in turn leads to production of MASs that are difficult to maintain and reuse Since Non-functional concerns are key criteria in determining the success of a software system, need to be addressed early in the software development lifecycle. As these properties interact with, or crosscut, many components and connectors in the architecture of a system. An aspect-oriented design approach appears to be a suitable solution for modeling them. Performance is most important non-functional concern. In our paper we focus presenting a framework that separates the performance aspects from other agenthood, functional and non-functional aspects. The use of our framework minimizes code replication, and increases maintainability and reusability of performance and agenthood concerns.

1. Introduction

Software architecture and design are artifacts that are produced early in development phase and gives an early solution decision for the given software requirement. So, they should be carefully designed because in later stages many bugs are identified and a lot of cost is consumed in fixing them [1]. In a distributed environment, software engineers face challenges of system reusability and maintainability as the agent applications grow in size and complexity. Many of the issues such as performance tend to crosscut several methods and classes, implementing other concerns of MAS's. This problem often leads to the poor separation of concerns which in turn produces agent systems that are difficult to maintain and reuse [3]. When usually we design and implement a system we modularize it into small units such as objects, modules and procedures. Decisions based on separation of concerns usually confine to functionalities. Some non-functional aspects such as performance (our focus), synchronization, security control, interaction or persistency are implemented as lines of code scattered over many software

units [3,16,17,18]. In this context, the aspect-based framework provides separation of Performance concerns among the different Agenthood properties and collaborative capabilities. The introduction of software agents in the object model poses other problems because many properties and capabilities of agents are intrusive and not orthogonal; therefore a disciplined approach is required for composition. As a consequence, a need arises for an agent framework, which supports the manipulation of these properties and capabilities directly from the very beginning of design in order to master the natural complexity of building agent-based object-oriented systems. Performance is one of the most commonly required non-functional quality of software [2].

1.1 Objects and Agents

AOP is the programming paradigm that realizes the principle of separation of concerns and lets the programmer focus on one aspect [4]. Aspect oriented design encourages modular description of software systems by providing support for cleanly separating the object's core functionality from its crosscutting concerns. Aspects are the units that modularize these crosscutting concerns and are associated with one or more objects. These are comprised of Pointcuts, advices and introduction. Join points are principal points in the dynamic execution of an object while Pointcuts are collections of join points; advice is a special method-like construct that can be attached to Pointcuts. Introduction is a construct that defines new ordinary member declarations to the object(s) to which the aspect is attached (such as, attributes and methods).

Central to the process of composing aspects and objects is the concept of weaver. Weaver is the component responsible for the deviating the normal control flow to an advice which activates when program execution point is at a join point. AspectJ [13] is a practical aspect-oriented extension to the Java programming language. AspectJ implements a weaver and provides support for programmers defining aspects.

1.2 Aspect Oriented Programming (AOP)

AOP is the programming paradigm that realizes the principle of separation of concerns and let the programmer focus on at one aspect at one time [4]. Aspect oriented design encourages modular descriptions of software systems by providing support for cleanly separating the object's core functionality from its crosscutting concerns. Aspects are the units that modularize these crosscutting concerns and are

associated with one or more objects. They are comprised of Pointcuts, advices, and introduction. Join points are principal points in the dynamic execution of an object. Pointcuts are collections of join points; advice is a special method-like construct that can be attached to Pointcuts. Introduction is a construct that defines new ordinary member declarations to the object(s) to which the aspect is attached (such as, attributes and methods). Central to the process of composing aspects and objects is the concept of weaver. Weaver is the component responsible for deviating the normal control flow to an advice, when program execution point is at a join point. AspectJ [13] is a practical aspect-oriented extension to the Java programming language. AspectJ implements a weaver and provides support for programmers defining aspects.

1.3 Overview of proposed Approach

Fig Fig. 1 shows that the design includes application specific design, performance design and therefore code for the design model, code for application specific design and code for the performance concern get mixed together when they are implemented. It is assumed that we manually obtain code from the design model. Then the source code is compiled, executed and analyzed. If necessary, the system is redesigned and the whole process is repeated.

The disadvantages this of the conventional approach brought are that (1) it is difficult to develop the design model because the designer has to all the software designs take into the account and the performance models at the same time, and (2) it is difficult to understand and maintain the program code

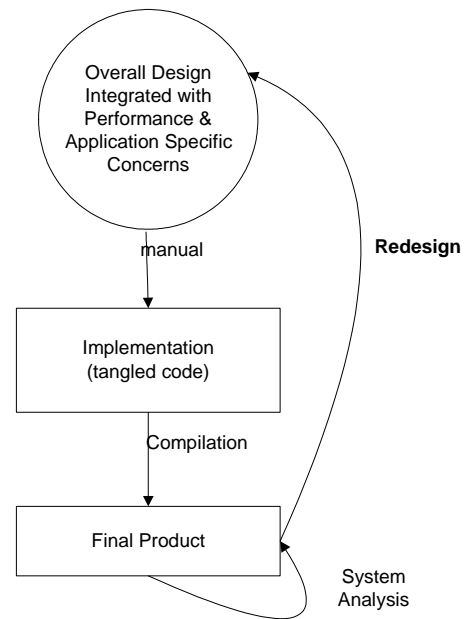


Fig. 1. Traditional Approach.

because several concerns are intermingled in the implementation code. Here AOP helps us to modularize the software by allowing us to express various aspects independently from functionality. In particular ,AOP enables us to model and design the performance aspect of software system during design phase.

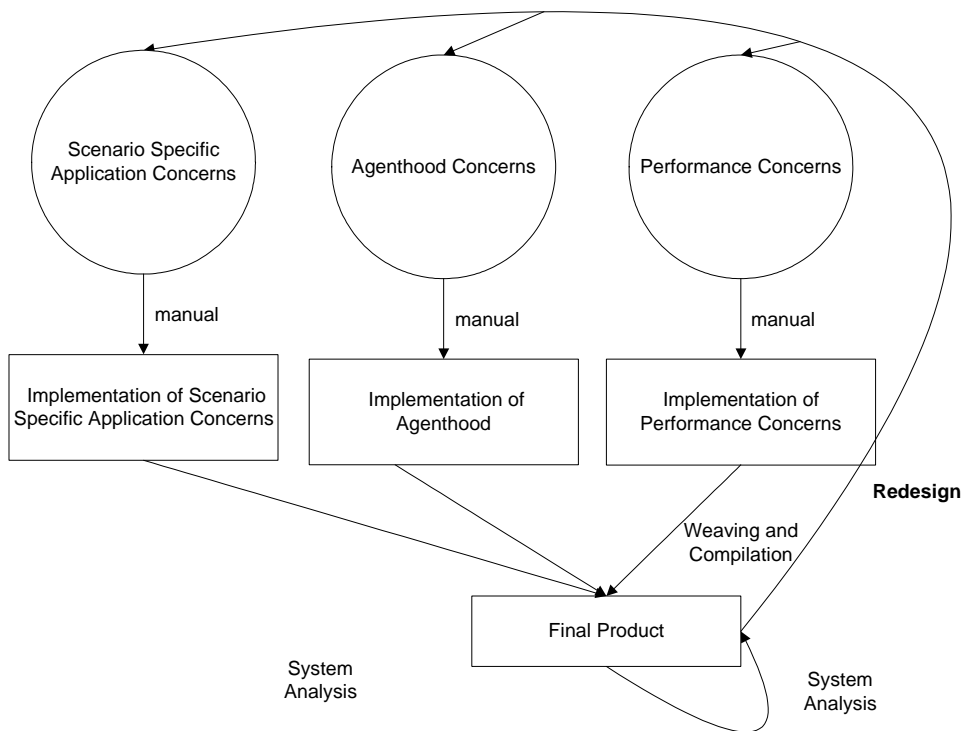


Fig. 2. Overview of Proposed Approach.

Fig.2 shows an overview of proposed approach. Here an aspect-oriented framework for agent-oriented software, which supports (1) Clear separation between performance specific concerns, other agents specific concerns and scenario specific concerns. (2) Aspects are used as unifying abstractions to capture the performance agent concerns which are hard to modularize with agent-oriented abstractions [2]. In the design phase, the performance model is set apart from the design model and the separation to the implementation model is maintained. In simple words, the design model for scenario specific application concerns and agenthood concerns has its own Java implementation code while the performance model has a separate AspectJ implementation. Later, these are all woven together using the AspectJ compile.

Based on this separation the AOP framework that has been identified as abstracted from instantiation in different agent applications. And from the study of a number of frameworks [5, 6, 9, 10, 11, 12, 13] as a consequence they can be applied to wide range of applications.

The remainder of this paper is organized as follows. Section 2 focuses on main concerns related to performance of a system. Section 3 gives a brief overview of crosscutting nature of performance concerns. In section 4 we propose our framework. Section 5 related works section 6 future work and conclusion.

2. Performance Concerns

Performance of the agent oriented software depends on several concerns which can be categorized as follows (1) Agenthood (2) Scenario Specific Functional Concerns (3) Resource Concerns (4) Workload Concerns (5) Scenario Specific non-functional Concerns.

2.1 Agenthood

Agenthood depicts and defines the essential features associated with agent concepts. it consists of the basic characteristics of the agent. These are shown in Table-1.

2.2 Scenario Specific Requirements

Scenario Specific Requirements are the functional requirements of the software that are specific to the application scenario. The separations of these concerns are required from the agenthood and other concerns. From the detailed study and research some important attributes and parameters are identified for performance concerns which are common to all agent based application scenarios. While there are some other parameters that are specific to each step of the execution for specific scenario. Both parameters are shown in Table 2.

2.3 Workload and Resources

Performance has a critical relation with workload and resources as the workload increases the performance of the system may tend to decrease and vice versa. The Basic

TABLE I
AGENTHOOD/AGENCY CONCERNS

CONCERN	DESCRIPTION
Mobility	Being able to migrate in a self-directed way from one host platform to another.
Interaction	The property to send and receive message to each other in MASs.
Autonomy	Goal-directedness, proactive and self-starting behavior.
Adaptability	Being able to learn and improve with experience.
Learning	Learning from own experience, its environment, and interactions with others.
Collaboration	Can work in concert with other agents in order to achieve a common goal.

attributes of the workload that affects the performance of the system are shown in table 2.

On the other hand when resources are in crease the performance increases and vice versa. If we observe it closely,

TABLE 2
PERFORMANCE PARAMETERS

Scenario Specific Parameters	General	Response time Host execution delay
	Each Step	Probability Repetitions Delay Operation Intervals Execution Time
Workload	Response time Workload Priority Execution Delay Population	
	Utilization Scheduling Policy Throughput Processing Rate Waiting time Context Switching Time Response Time Priority Range Capacity Perceptibility Access time	

it can be found that the most performance effecting attributes of resources shown in table 2.

2.4 Scenario Specific Non-Functional Concerns

Performance is also dependent upon the Scenario Specific non-functional requirements like security and reliability etc. And there is always a deep interdependency between performance and these requirements.

3. Crosscutting Performance Concerns

Although OO frameworks are essential to the development of agent-oriented software, they are intrusive and lead to a poor system modularization [3, 16], which in turn decreases the system maintainability and reusability. The OO frameworks impose architectural restrictions on the agent design, which lead to the tangling and scattering of the performance and other concern [3, 16, 18, 21]. In order to make agents efficient (performance), developers usually have to change their agents design to extend performance-specific classes, implement abstract methods of those performance

functionality and collaborative activities are amalgamated to performance methods (problem-1). The agent and role classes also need to hold an explicit reference to performance elements as attributes (problem-2). These classes have additional methods to manage these elements (problem-3). In addition, several methods have performance code in them in order to decide how the agent should act efficiently in the specific environment (problem-4). As a result, this code is replicated on various methods of plan, role, and agent type classes.

All these problems decrease the system reusability and maintainability, since adding or removing the performance

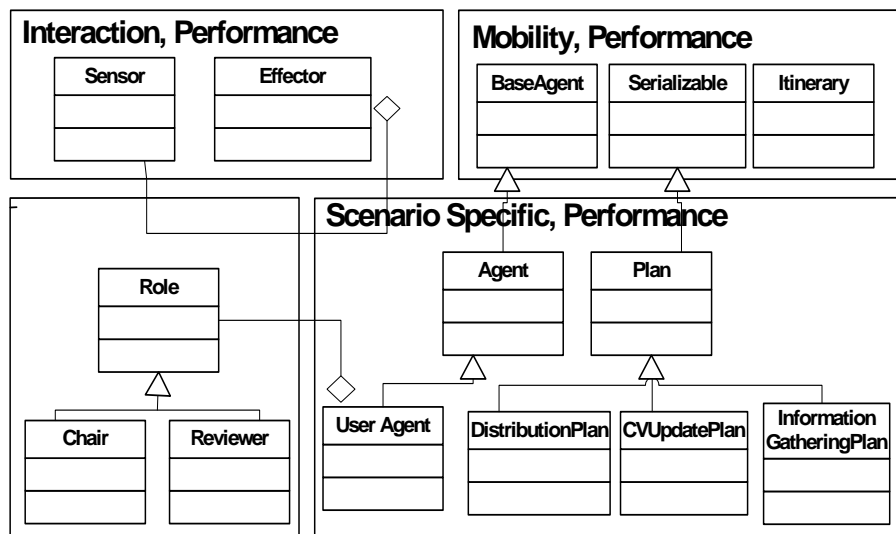


Fig. 3 . Crosscutting Performance Concerns

classes, implement performance-specific interfaces, and invoke explicitly performance methods in the classes implementing other agent and other scenario specific concerns.

Fig. 3 shows a partial representation of a multi-agent system [10], which will be also used in Section 4 to show the applicability of our proposal. For simplification, it only shows some important classes, the others essentially follow the same pattern; we also omit the classes related to learning, adaptation, and autonomy. Each set of classes, has the main purpose of modularizing a specific agent concern, namely interaction, collaboration, mobility and basic concerns. However, note that the performance concern crosscuts classes implementing other agent concerns has a huge impact on the basic agent structure and on the collaboration and interaction designs.

There are classes that represent the agent types and roles; they extend the abstract Base Agent (specific to the underlying MAS Platform) class to incorporate the performance capabilities. However the use of sub classing results in both code replication and code tangling; the basic

code from classes requires invasive changes in those classes. Note that we cannot find a more modular solution even if we try to either re-factor the object-oriented solution presented in Fig. 3. This problem happens because performance is a crosscutting concern independent of the object-oriented decomposition used [3, 16]

4. Proposed Framework

This section present the proposed framework that supports the aspect-oriented modularization of the performance concern, the framework is presented in terms of its software architecture (Section 4.1).

4.1 Framework Architecture

Fig. 4 presents the architectural components of the proposed framework. The architecture modeling is based on the aSideML language [20], which is a UML extension for representing aspects at different levels of abstraction. Architectural aspects are UML components represented as diamonds. Fig. 4 also illustrates the component interfaces.

Each interface is displayed as a small circle with the interface name placed next to the circle. Each architectural component has one or more interfaces. The interfaces are categorized in two groups: *normal interfaces* and *crosscutting interfaces* while normal interface provides services to other components only. Crosscutting interfaces specify when and how an

adaptation, mobility etc.

- The agent platform component represent the under lying MAS platform.
- Workload and resources are supporting components for measuring and reporting performance.
- The IInformatioGathering interface of Performance

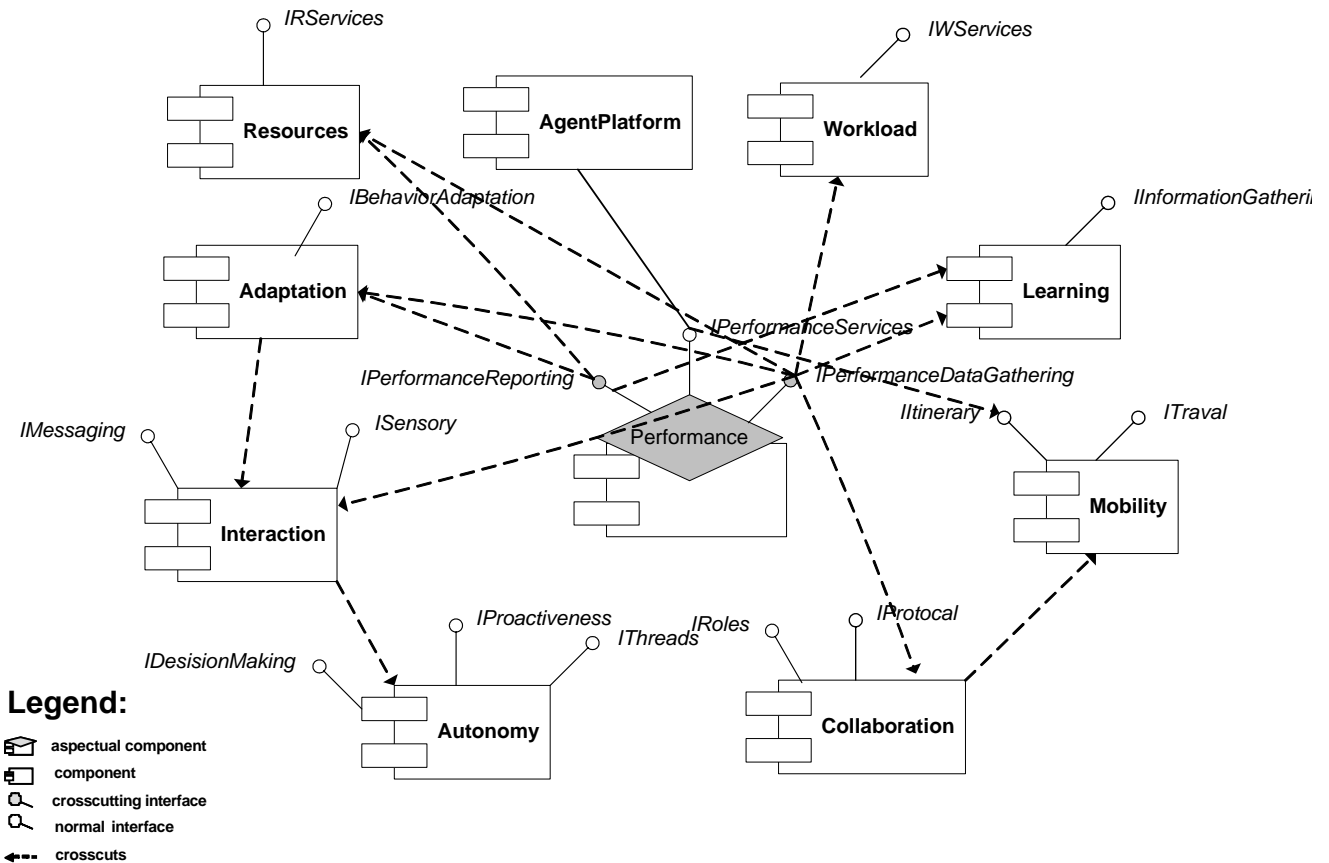


Fig. 4. Proposed Performance Architecture

architectural aspect affects other architectural components. The purpose of crosscutting interfaces is to modularize parts of the performance concern, which usually crosscut other concerns in traditional kinds of decomposition, such as object-orientation

Our architecture follows an aspect oriented architecture [10], which is composed of ten kinds of components.

- Performance Component that is the aspectual component, the notion of aspectual component also called architectural aspect [10] modularizes crosscutting concerns at the architectural level. In this way, performance component separates the performance concerns from other agent and application concerns including interaction, collaboration, mobility etc.
- The components that represent the other agent concerns are collaboration, interaction, learning, autonomy,

component is used to interact and get performance related data from the Workload and Resources components.

- IperformanceReporting interface of the Performance component reports the information related to performance to Learning component directly.

The Performance aspectual component has a crosscutting interface IperformanceServices that provide the Performance related services to the AgentPlatform component. Note that the performance aspectual services are related to more than one architectural component, representing its crosscutting nature (fig. 4.). The Performance aspectual component realizes two crosscutting interfaces, since it can crosscut multiple agents components in different ways. These interfaces crosscut either internal elements of an agent component or other interfaces.

5. Related work

Kendall et al [7] describes the application of aspect-oriented programming to implement role models. This approach is used to represent the different collaborative capabilities (roles) an agent can have. In fact, we have followed their guidelines to implement agent's collaborative aspects. However, their work does not deal with Agenthood properties, which we believe are the main source of agent complexity. In this sense, our paper presents a unified framework for dealing with performance and collaborative capabilities and Agenthood properties, and their interrelationships.

Research in aspect-oriented software engineering has concentrated on the implementation phase. A few works have presented aspect-based design solutions. In addition, since aspect-oriented programming is still in its infancy, little experience with employing this paradigm is currently available. To date, aspect-oriented programming has been used to implement generic aspects such as persistence, error detection/handling, logging, tracing, caching, and synchronization. However, each of these papers is generally dedicated to only one of these generic aspects. In this work, we provide an aspect-based design model which: (i) handles nonfunctional (e.g. performance), Agenthood-specific aspects as well as generic aspects and (ii) encompasses a number of different aspects and their relationships.

6. Conclusion and Future Work

The facet of our strategy is to separate and make transparent the performance concerns so that changes in the performance concern have no impact on the implementation of the other agent concerns. This paper presents an aspect-oriented framework that modularizes the Performance concern crosscutting several classes and methods of a software agent. The main contribution of this work is a design model, which provides a unified framework for introducing Agenthood aspects to the object model. Moreover, this agent model: (i) incorporates flexible facilities to build different kinds of software agents, (ii) encourages the handling of each of the Agenthood aspects separately, (iii) provides explicit support for disciplined and transparent composition of non-functional properties and collaborative capabilities in complex software agents, and (iv) allows the production of agent-based software so that it is easy to understand, maintain and reuse. It allows not only the specification of the basic Performance behaviors, but also the specification of the agenthood characteristics and application specific functional and non-functional concerns at the same time. Note that the use of our framework results in fewer lines of code since agent classes do not need to incorporate performance code.

7. References

- [1] Daesung Park and Sungwon Kang, Design Phase Analysis of Software Performance Using Aspect-Oriented Programming, Information and Communications University Munji-ro 119, Yuseoung-gu, Daejeon, 305-714, Korea.
- [2] Kendra Cooper, Lirong Dai, Yi Deng, Jing Dong "Modeling Performance as an Aspect: a UML Based Approach" 2003.
- [3] N. Ubayashi, T. Tamai. Separation of Concerns in Mobile Agent Applications. Proceedings of Reflection 2001, LNCS 2192, Kyoto, Japan, September 2001, Springer, pp. 89-109.
- [4] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, et al., "Aspect-Oriented Programming," Proc. ECOOP, Springer-Verlag, 1997.
- [5] M. Katara and S. Katz, "Architectural Views of Aspects," Proc. 2nd International Conference on Aspect-oriented software development, pp.1-10, March 17-21, 2003.
- [6] F. Bellifemine et al. JADE: A FIPA-Compliant Agent Framework. In Proceedings of the Practical Applications of Intelligent Agents and Multi-Agents, April 1999; pp. 97- 108.
- [7] E. Kendall. "Agent Roles and Aspects". ECOOP Workshop on Aspect Oriented Programming, July, 1998.
- [8] N. Noda and T. Kishi, "On Aspect-Oriented Design: An Approach to Designing Quality attributes", APSEC 1999.
- [9] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni,
- [10] "Model-based Performance Prediction in Software Development: A Survey," IEEE Trans. SE, 30(5), May 2004.
- [11] A. Garcia. From Objects to Agents: An Aspect-Oriented Approach. PhD Thesis, Computer Science Department, PUC-Rio, April 2004.
- [12] J. Gray, T. Bapty , S. Neema, D. C. Schmidt, A. Gokhale, and B. Natarajan, "An Approach for Supporting Aspect-Oriented Domain Modeling," Proceedings of 2nd International Conference on Generative programming and component engineering, pp.151-168, Erfurt, Germany, Sept. 22-25, 2003.
- [13] FIPA. Foundation for Intelligent Physical Agents. <http://www.fipa.org>.
- [14] JADE Tutorial and Primer, Jean Vaucher and Ambroise Ncho, Dep. d'informatique Université de Montréal, September 2003 updated april 2004.
- [15] An Introduction to Software Agents, Jeffrey M. Bradshaw
- [16] Agent Technology: Enabling Next Generation Computing, A Roadmap for Agent Based Computing, Michael Luck, Peter McBurney, Chris Preist
- [17] A. Garcia et al. Separation of Concerns in Multi-Agent Systems: An Empirical Study. In: "Software Engineering for Multi-Agent Systems II", Springer, LNCS 2940, April 2004.
- [18] A. Garcia, C. Lucena, D. Cowan. Agents in Object-Oriented Software Engineering. Software: Practice and Experience, Volume 34, Issue 5, April 2004, pp. 489-521.
- [19] A. Garcia et al. Engineering Multi-Agent Systems with Aspects and Patterns. Journal of the Brazilian Computer Society, Number 1, Volume 8, July 2002, pp. 57-72.
- [20] SAGE: Scalable Fault Tolerant Agent Grooming Environment, <http://sage.niit.edu.pk>
- [21] C. Chavez. A Model-Driven Approach to Aspect-Oriented Design. PhD Thesis, Computer Science Department, PUC-Rio, April 2004, Rio de Janeiro, Brazil.
- [22] S. Deloach et al. Multiagent Systems Engineering. International Journal of Software Engineering and Knowledge Engineering, 11(3):231--258, 2001.
- [23] [22] M. Shaw, D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall (1996).