5th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence and Informatics
January 25-26, 2007 ░ Poprad, Slovakia

Using Agents and Aspects for Improving of Reliability in Open Design Architecture[1]

Miroslav Beličák, Marcel Tóth

Department of Computer and Informatics, Technical University of Košice
Letná 9, 051 20 Košice, Slovakia, {Miroslav.Belicak, Marcel.Toth}@tuke.sk

Abstract: Contemporary software systems generally suffer from problems of adaptation to new requirements. This is particularly true for information systems, where changes are regular. This article describes utilization of agent technologies (software agents) for improving security and runtime modifiability of information systems (IS) based on proposed open design architecture (ODA). Main characteristics of ODA are introduced together with description of its runtime environment. Emphasis is on exploiting of artificial intelligence agent (AI agent), which facilitates smooth recomposition of running system, without need for user disconnection or stopping the system. In presented architecture, software agents have also important use for managing security of IS based on ODA. Principal advantage of ODA originates in joint IS representation consisting of system's design linked to its binary form. This enables extracting and changing of design embedded in IS, which is expressed as changed properties of (possibly) running IS. Dynamic readjustment of system's implementation to new design is provided by aspect oriented approach and mentioned AI agents.

Keywords: design, artificial intelligence agent, software agent, aspect, system, information system, run-time environment, information system architecture

1 Introduction

Agents provide software designers and developers with a way of structuring an application around autonomous, communicative components. In this sense, they offer a new and often more appropriate route to the development of complex computational systems, especially in open and dynamic environments [10,8]. The concept of an agent has found currency in a diverse range of information technology. Especially, agent-oriented software engineering is being described as

[1]This work was supported by:
VEGA 1/2176/05 – Technologies for Agent-based and Component-based Distributed Systems Lifecycle Support
VEGA 1/4073/07 – Aspect-oriented Evolution of Complex Software Systems

a new paradigm for the research field of software engineering [16]. This article introduces software agents (SW agents) and artificial intelligence agents (AI agents) to the operation and supervision of information systems (IS) based on newly proposed architecture called ODA. Core idea supporting this work is endeavor to create pattern for development of IS containing all information needed to its modification, scalability, robustness etc. In this manner, ODA prescribes joint representation of IS, design together with executable parts. For this purpose, ODA divides information system to aggregate functional parts called services and indivisible functional units called binary micro-components. Complete IS in ODA is represented by the set of micro-components and a number of design diagrams determining higher-level units (services), control-flow, data-flow, security and other aspects of IS. ODA ideas thus successfully combine the principles of other IS architectures (SOA, MDA and CBA) [1]. Agents as described in this article, seem to be suitable solution for securing of ODA runtime environment (SW agent) and for controlling and optimizing of non-deterministic interaction with users (AI agent).

2 Agents

Agent-based applications and agent systems represent a very robust and theoretically well funded technological paradigm [5]. Despite this, they are not yet widespread at all due to many reasons, one of which is the heavy programming work that still has to be done in order to get efficient agent systems in particular from the point of view of the performance and integration with other applications.

In general, the agent is a concrete entity (a piece of software) that sends and receives messages, while the service is the set of functionality that is provided [17]. We can divide agents into two basic groups:

- Software agents (SW agents), and
- Artificial Intelligence Agents (AI agents).

These groups will be characterized in next sections.

Software agent is adaptive and intelligent software component that acts autonomously on behalf of it's users, access and analyze information and services, is situated in an environment and react to changes in the environment, has a set of objectives; and cooperate and coordinate it's activities in order to accomplish a goal. [14,12,18,7,6].

Another definition of software agent [15]:

A *software agent* is a program, executed at a given place, characterized by:

- autonomy – agents process their work independently without the need for human management

- communication – agents are able to communicate with one another, as well as with humans
- learning – agents are able to learn as they react with their environment and other agents or humans

Important properties of software agent are also:

- it gets nearer to the abstraction of actor than objects
- it is higher-level component abstraction of system modeling
- for its design and implementation it is possible to use object-oriented approach, or design patterns respectively

The additional tiers of behaviour such as learning, code mobility, etc. could be added in direct way to the systems implemented with agent technology.

Artificial Intelligence agent (AI agent) has against SW agent several of next properties, especially: Adaptability, Intelligence, Rationality, Transparency and accountable [13].

'*AI agent's state must be formalized by knowledge (i.e., beliefs, goals, desires, intentions, plans, assumptions) and be able to act on this knowledge. It should be able to examine its beliefs and desires, form its intentions, plan what actions it will perform based on certain assumptions, and eventually act on its plans. Furthermore, intelligent agents must be able to interact with other agents using symbolic language. All this sounds like a model of rational human thinking — but we should not be surprised*' [13].

3 Brief Description of ODA Architecture

Every information system designed in context of ODA consists of two main parts:

- Application Logic (AL)
- System Design (SD)

Under the term application logic (AL) we intend specific group of services offered by particular IS, where the service is executable part of information system, covering exactly one functionality of system. The service takes input data from user of IS (or application representing the user) and produces output data (principle similar to web services [3], but unlike web services, AL is located in local IS services repository[2]). Every such service is responsible for designated area (e.g. Various types of database operations, calculations, security controls, data processing, etc.) and consists of interconnected and mutually interacting components called binary micro-components (BMC). BMC's have the form of individual executable unit (dynamically linked library satisfying defined conditions, ...) representing functionality fragment. This standalone unit (BMC) can be one class[2], implementing specified interface for example. Common interface allows communication with environment (ODA and other BMC's and services in ODA) and also allows the class to be the building block of services and finally running the service.

Design of IS comprises of three separated sub-designs, which have the form of several diagram types. The first one, Application Logic Design – describes AL (Figure 1, Figure 2).

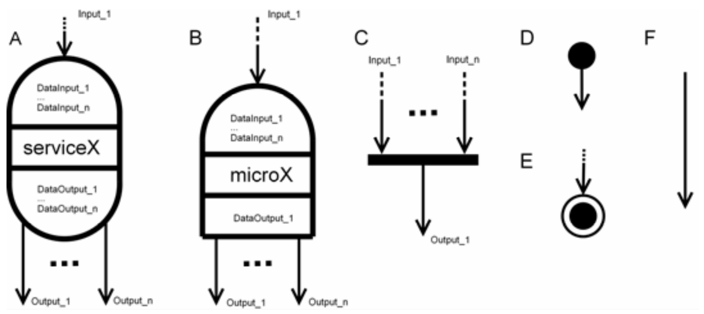

Figure 1
Elements of Application Logic Diagram

Element **A** represents the service of IS, where inputs are represented by arrow aiming from the top and outputs by arrows aiming at the bottom. The same is true for element **B**, which is called binary micro-element (BMC). It's meaning is analogous to service, the only difference is that BMC has no visible internal structure (it is not decomposable in application logic diagram, so it is primitive unit of functionality) and has output consisting of only one sub-value. Both services and micro-elements have only one output value (structured to sub-values in any fashion), however more output arrows means copying and possibly transforming of this value to more recipients in parallel branches of execution. Visual appearance of described elements is divided to three vertical levels: first (top) level specifies input sub-values relevant for element under consideration, name of element resides in the middle and output sub-values are in the last (bottom) part. Element marked as **C** indicates synchronization of more inputs to one output. It includes control and data synchronization, control synchronization

2 Of course, services can be placed in distributed environment – arbitrarily in Internet.
2 If considered object oriented paradigm.

as waiting for all inputs and data synchronization as transformation of several input values to one output value. **D** and **F** are only representing start point and end point of diagram of one service. Marks used for **D**,**F** are taken from state-chart diagram because of comprehensibility. Last element (simple arrow) marked as **F** represents flows of control and data between other elements of the diagram.

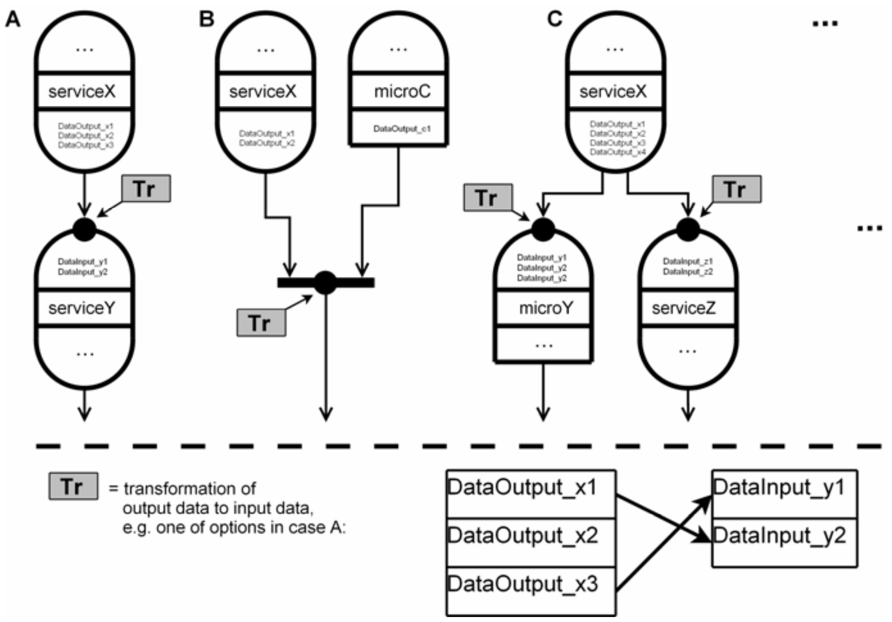

Figure 2
Possible interconnections of Application Logic Diagram Elements

Possible connections between services or BMCs are shown in the Figure 2. Three cases are depicted: ‘1 to 1’ (in the case **A**), ‘n to 1’ (in case **B**) and ‘1 to n’ (in case **C**). ‘Etc.’ symbols for subsequent cases mean various combinations of different input and output elements for these three primary cases. Restriction in current version of diagram semantics is that all branches coming out from ‘1 to n’ branching must be synchronized in one ‘n to 1’ branching, which is possible only with synchronization element (as in the case B). Connection of elements means input/output value passing (except of control passing, which is also the meaning of used arrow). Since connected elements doesn't need to be fully data compatible, transformation is needed to adapt output sub-values to input sub-values of the second element. The places where this is taking place is marked by black ring and labeled by block ‘Tr’ in the Figure 2. In the lower part of Figure 2, two interconnected parts are shown as tables. Each table represents input or output value consisting of its sub-values displayed as rows. The tables in the picture have only 1 column for simplification and belong to case **A**, where three output sub-values (x1,x2,x3) are transformed to 2 input sub-values (y1,y2). Transformation in this case is quite simple: only reordering of sub-values and omitting of output x2, but this doesn't need to be true for complex values. The table has only 1 row (1 output sub-value) in case of BMC - 1 output is similar to number of outputs of function or method in many programming languages. Columns in full version of this table are: name of the input/output variable, data type of the variable and human readable description. Interconnection of input and output sub-values is predefined for service and is designated in time of designing of the service, probably in graphical integrated development environment (IDE). Similarly to function in procedural programming or method in object oriented programming, the service or BMC can have zero inputs or outputs.

The second one – Security design – designs security aspects – associating user roles with individual services in AL. The third sub-design describes organization structure (hierarchy) of respective services (based on types of service functionality) – Service Hierarchy Diagram. User-designer thus has the possibility of extracting systems design in case of required change. After design extraction, specific integrated development environment (IDE for modifications of design in ODA) can be used to apply required changes – even without breaking of IS's execution and disconnection of users (no consultations with previous designers of system are necessary, because AI agent manages relevant changes). Security considerations of runtime environment and all other security concerns are the responsibility of security manager represented by software agent (SW agent). On the other hand, AI agent's task is to tune-up performance and efficiency for users of IS. Its work includes profiling and setting priorities on behalf of users. Figure 3 shows main parts of ODA.

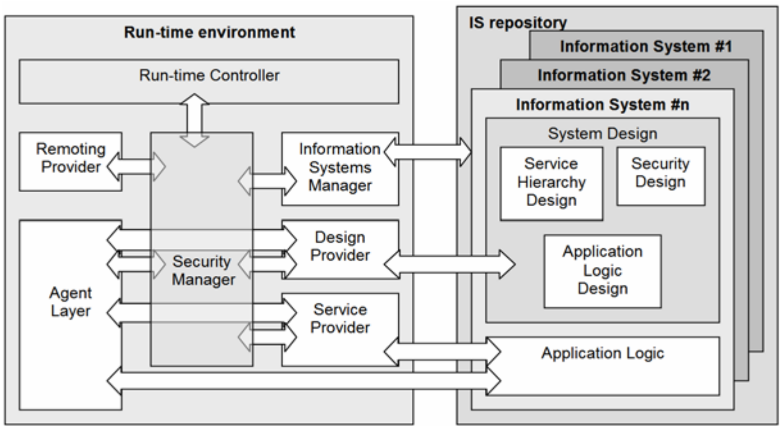

Figure 3
Open Design Architecture

If change in IS design (and thereby in whole structure of IS) is requested, current design of IS is sent to designer-user through ‘Design Provider’. After potential modifications by designer, the design is sent back to ‘Design Provider’ (through ‘Remoting Provider’ and ‘Security Manager’), where modified design is transcribed to original design of IS. This change can be accomplished in running IS without disconnection of IS users. Changes in running system can bring many problems, like security risks, loss of data or inefficiency of system. This is the point, where properties of agents show handy.

According to presented ODA principles, ODA combines properties of SOA [3] – regarding services, MDA – regarding design of IS independent from target platform of IS and CBA – services consist of binary micro-components.

5 Utilizing Agents and Aspects in Run-time Environment of ODA

5.1 Utilization of Software Agents

Software agent is in proposed architecture put into effect in the component 'Security Manager'. This software agent should identify and authenticate service consumers. Moreover, it should secure:

- execution environment of ODA from unauthorized accesses
- communication between service consumers and execution environment
- communication between remote information systems based on ODA (communication between remote execution environment) – Figure 4
- communication between designer and execution environment in case of:
  - ♦ reviewing or changing of IS structure by designer
  - ♦ adding/removing of IS to/from IS repository by designer
- validate and verify security credentials of individual service users

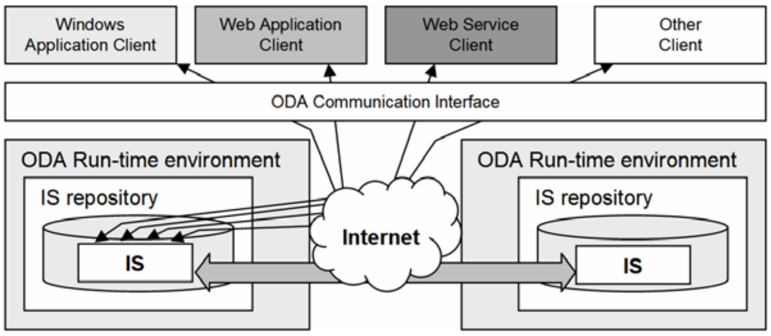

Figure 4
Remote co-operating IS based on ODA; clients access IS through communication interface

The software agent should be simply responsible for everything regarding security of execution environment. One of application of software agent under review is its use for assembling of new services in compliance with design modifications.

5.2 Utilizing Artificial Intelligence Agents

AI agents will be used for optimization of individual services execution performance as viewed by users. This will also influence efficiency of whole execution environment for ODA information systems. Every user of such IS is characterized by following information:[1]:

- number and duration of log-ons to IS
- Interest of user in particular service (user preferences)
- number of service usage
- usual time (or time interval) of service usage in periodicity context of days, weeks, ...
- order of service usage by user
- intervals of execution for individual services
- input/output parameters of consumed services (size, type, …)
- intervals of execution for various service parts (sub-parts, or micro-components) – knowledge of IS structure is required
- priorities of individual users

All listed evidence constitute so-called **user profile**. The term user means external entity, that is consuming services of relevant ODA based IS. It is analogy of actor from use-case diagram of modeling language UML. Such user can be any physical user, or external IS exchanging information with IS under consideration.

Especially interesting is last parameter of user profile – **priority of user**. Every role[2] of user in the system will have default priority of type 'normal'. If required, user authenticated to do so can increase his priority for the service (before using the service) – priority system is thus similar to priority in e-mail communication[3]. Thereby, user signals to the system, that consumed service is required to speed up and system will use actual resource allocation strategy to fulfill user requirements (for simple example, system can prefer higher priority user among users with the same role). Roles, in which individual consumers emerge in the system will have

[1] If ‘security credentials’ such as login, password are omitted in this enumeration
[2] User is assigned his role by execution system in time of authorization upon user's credentials (login and password).
[3] Priority can also have integer value from chosen interval, where higher number means higher priority and vice versa.

precise hierarchy[4]. Of course, frequence of priority amendment for the user will be monitored by system and the number of such changes can be limited for some time period (and for user or role). This prevents abuse of the priority technique.

Based on user profile, AI agent can prefer chosen users. Example – particular user often consumes some service of short duration. This even happens in regular intervals. However, the service consumed is also requested by another user, which doesn't access services very often, but execution of services requested by him is considerably resource (time, memory, ...) consuming, because he nearly always requests operations on extensive data. AI agent monitoring both users and creating their profiles will decide that most of the time, the first user is preferred. But because this is AI agent (which can be non-deterministic), his decisions can change in time, possibly converging to ideal solution.

It is necessary to perceive, that in case of user profile, mostly pure-statistical data determine the result. On the basis of these data, AI agent will alter execution of respective services (or their parts) and thus performance of whole runtime system executing the IS. In view of non-determinism of AI agent operating on statistical data, the system needs shield against bad decisions of agent. The shield is included in the principle of ODA - the AI agent is only advisory for 'Service Provider' component, so it can only help and not harm (it cannot destroy data). Using the information sent from AI agent, 'Service Provider' as part of ODA runtime environment will accelerate or slow down execution of given service. As was described, decisions of AI agent don't influence very execution of service or data dispatching, only the speed of execution is manipulated. Utilization of AI agent is in our opinion suitable experiment for combination of standard IT technologies with chosen abilities of artificial intelligence. Work of AI agent in ODA is depicted in the Figure 5.

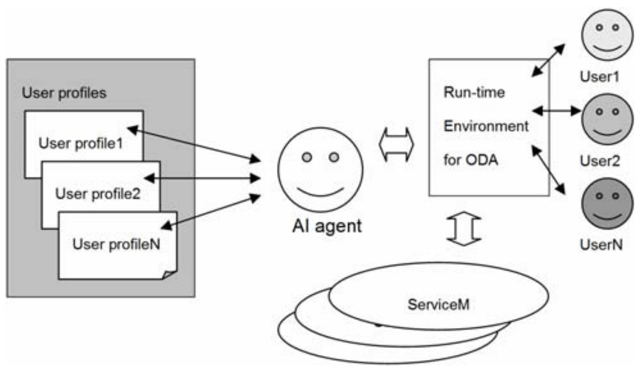

Figure 5
Utilization of AI agent for profilation of services for individual clients of ODA-IS

[4] It can be imagined as the shape of tree, where the root is administrator and lists are roles representing lowest competences of particular community (e.g. company) for which is IS built.

AI agent with following properties is needed for this purpose:

• autonomy – ability of independent decision making, in this case for user preference among the others

• planning – ability to use current and past information (e.g. user profiles) for scheduling execution or profilation of services in advance.

• intelligence – ability to determine variables for the decision. This property opens the possibility of knowledge base utilization (service structure, user profiles, ...)

5.3 Utilizing Aspects

There is no space for explaining principles of aspect oriented programming (AOP) in this article, so more information on AOP can be found in [9,4,2,11].

Figure 6 briefly introduces the place for aspect oriented programming (AOP) in development of systems in ODA. Figure needs to be perceived as description of direct steps leading to running system, not as a steps of system's life-cycle or guide to creating system in ODA. For example step 2 (service definition) is regarded as design step and should precede step 1 (compilation - part of system's implementation) in progress of system's design. All steps and their purposes are commented in next paragraphs.

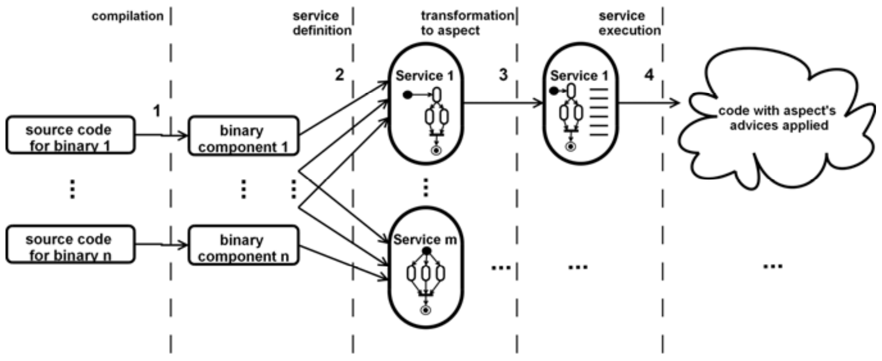

Figure 6
Utilization of aspects in ODA

First step (dashed line marked 1 in Figure 6) is compilation, standard step in programming of systems, transition from source code to executable version of the system. In the figure, this represents transition from source code of binary micro-components (BMCs, as smallest part of system in ODA) to their binary form, compiled binary component. These BMCs have inputs, outputs, description of their functionality and they are stored in common repository supplied by ODA runtime environment (possibly, they are only .dll libraries in file-system).

Second step named ‘definition of service’ means assembling of services from binary BMCs and/or other services. Practically, it means creation of application functionality design diagrams in supporting design environment (under development). After this step, expected system should be fully designed and implemented. Idea of ODA is to keep system's functionality together with it's design. This is achieved by only keeping design (with aspect code added later), counting on functionality in common repository (in BMCs). Next steps (3-transformation to aspect and 4-service execution) are only ways of bringing working system.

Third step ‘transformation to aspect’ is most important, because it prescribes how will be design transformed to code. Part observable in Figure 6 is, that created aspect is stored with design of service and re-created after each change in design of service.

And finally fourth step ‘service execution’ designates execution of service on demand of user, this includes process of weaving, described later in this article.

6 Future Work

Since some key issues of ODA were identified in current work on this architecture (e.g. Details of implementation of AI agent, efficiency and dynamism of recomposition mechanism), it is crucial to focus on these issues and to elaborate solutions in detail. It is particularly important to consider all important alternatives. It is likewise necessary to augment diagram notation specification to the level, on which diagrams will be sufficiently expressive to fully describe arbitrarily complex systems. Future research will show whether this will be accomplished by using brand new diagram notation, by exploitation of some of existing standard diagram notations (e.g. From the language UML), or just by improving notation of already proposed diagrams such as ‘service hierarchy diagram’, ‘application logic diagram’, ...

In this work, usage of aspect oriented principles was nominated as recomposition mechanism for information systems based on ODA. This doesn't represent the only solution for recomposition mechanisms. Attempting to find other ways of recomposition (e.g direct use of OOP or compiler principle) will be the necessary step to prove execution efficiency and comprehensibility of chosen solution for designer/programmer. Last but not least, long term aim for future research and development in the area of ODA is implementation of case studies and prototype applications. These pilot implementations can (and hopefully will) show real advantages of ODA, which will retain viability of this new approach to IS architectures.

Conclusions

New type of architecture called open design architecture (ODA) was briefly introduced and its properties were explained. Proposed solution's idea is to interconnect functionality and design of the system, that allows smoother use and development. We presented the possibility of including an agent technologies in operation of execution environment for IS based on ODA. We showed utilization of software agent for communication and security activities of run-time ODA execution environment. We have also shown possible exploitation of AI agent for improved performance of before-mentioned execution environment and for profiling of services for individual users.

References

[1] Belicak M., Paralič M., Havlice Z.: Distributed Service-Oriented Information Systems with Open Architecture, in Proc. of ECI 2006, 7th International Scientific Conference, Slovakia, September 20-22, 2006, pp. 8-12

[2] Cytron R., Leavens G. T., editors. FOAL 2002: Foundations of Aspect-Oriented Languages (AOSD-2002), March 2002

[3] Dragoni N., Gaspari M.: Integrating Agent Communication Languages in Open Services Architectures, Technical Report UBLCS-2003-12, Department of Computer Science, University of Bologna, Italy, 2003

[4] Filman R. E., Friedman D. P.: Aspect-oriented Programming is Quantification and Obliviousness. Workshop on Advanced Separation of Concerns (OOPSLA 2000), Oct. 2000

[5] Grosso A., Gozzi A., Coccoli M., Boccalatte A.: An Agent Framework Based on the C# Language and the CLI, in Proc. of 1st Int.Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Computer Vision and Distrib. Computing, Czech Republic, Feb. 6-8, 2003

[6] Chira C.: Software Agents, IDIMS Report, February 21, 2003

[7] Jennings, N. R.: On agend-based soft. engineering. Artif. Intelligence, 2000

[8] Jennings N. R., Wooldridge M.: "*Agent-Oriented Software Engineering*," in Handbook of Agent Technology, Bradshaw, J., Ed.: AAAI/MIT Press, 2001

[9] Kiczales G., Lamping J., Mendhekar A., Maeda C., Lopes C.: Aspect-oriented Programming. Technical Report, February 1997

[10] Luck M., McBurney P., Shehory O., Willmott S.: Agent Technology: Computing as Interaction, AgentLink, 2005

[11] Marjan Mernik, Tomaz Kosar, Viljem Žumer: A Note on Aspect, Aspect-oriented and Domain-Specific Languages. Acta Electrotechnica et Informatica, FEII TU Košice, Slovakia, 5(1):1–8, October 2005

[12] Nwana, H. S. (1996). "Software Agents: An Overview." Knowledge Engineering Review 11(3): 1-40

[13] Object Management Group, Agent Platform Special Interest Group: Agent Technology (Green Paper), OMG Document agent/00-09-01, 1 Sept., 2000

[14] Pancerella C., Unruh A.: Software Agent Communities, in Proc. of the 32nd Hawaii International Conf. on System Sciences 1999, p. 8054, IEEE

[15] Rykowski J., Cellary W.: Virtual Web Services - Application of Software Agents to Personalization of Web Services, in Proceedings of ICEC'04, Sixth International Conference of Electronic Commerce, ACM 2004

[16] Tveit A.: A Survey of Agent-Oriented Software Engineering. Report, Norwegian University of Science and Technology, May 2000

[17] Walton Ch. D.: Model Checking Multi-Agent Web Services, Centre for Intelligent Systems and their Applications, University of Edinburgh, UK

[18] Wooldridge, M.: Intelligent Agents, The MIT Press, 1999