The Synthesis Stage in the Software Agent Development Process

Fernando Alonso[1], Sonia Frutos[1], Loïc Martínez[1], F. Javier Soriano[1]

[1] Facultad de Informática, Universidad Politécnica de Madrid,
28660 Boadilla del Monte (Madrid), Spain
{falonso, sfrutos, loic, jsoriano}@fi.upm.es

Abstract. In most existing software agents methodologies, system analysis is dependent on an agent-oriented, object-oriented or knowledge-based design paradigm. This simplifies the complex transformation of the conceptual model produced during analysis into the physical model output at design time. We, like other authors, believe that the conceptual model has to be conceived as independent of the design paradigm and that the physical model should be driven by the solution, both models leading to very different conceptions of the problem. In this paper we present the SONIA agents development methodology that includes a transitional synthesis stage between analysis and architectural design that mends the break between the construction of the two models.

1. Introduction

The software agent development process, and generally any software development process, can be viewed as the application of three transformations [1] (Fig. 1): requirements in the application domain are transformed into a conceptual model of the problem (T1- *Analysis*). The conceptual model is transformed into a physical model that represents the software product properties (T2-*Design*). And the physical model is transformed into a computable model, the program (T3-*Implementation*).

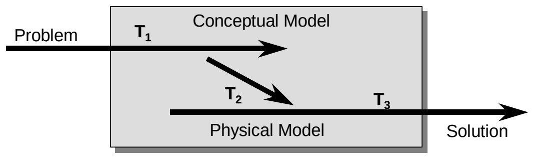


Fig. 1. The essential software process

The conceptual and physical models are two very different ways of looking at the problem, and its implementation involves a drastic change in the use of processes, methods and tools.

Additionally, there is no perfect method for either model [2]. The current trend is, therefore, to integrate different methods, tools and techniques, using whichever is the best in each individual situation. This raises the analysis dilemma: developers have to choose the best suited techniques for each problem. To make this decision, developers have to analyse the problem and choose a method of analysis before they are really familiar with the problem. If the chosen technique uses design terminology, then the problem-solving paradigm has been preconditioned, and this paradigm could turn out not to be the best suited when the problem has been analysed in detail. In conclusion, a good methodology should not force a given architecture upon developers from the very beginning. It is the outcome of the system specifications analysis that should point developers towards the best suited architecture for solving the problem [3].

By contrast with this idea, most agent methodologies propose design paradigm-dependent analysis to elude these problems and bridge the gap between the implementation of the two models (for example, Tropos [4], Gaia[5], Prometheus [6], MAS-CommonKADS [7]).

In this paper, we briefly describe a methodological approach that defines an architecture-independent generic analysis model, including, as the first design phase stage, a synthesis stage (which is the paper's core), that smoothes the step from the conceptual to the formal model. In Section 2, we describe the structure of the proposed SONIA methodology. Section 3 details the synthesis stage, and Section 4 states the conclusions on the advantage of this approach.

2. SONIA Methodology

Based on research and development efforts in the field of AOSE (Agent Oriented Software Engineering), we think that an agent-oriented development methodology should have the following features [8]: (i) it should not condition the use of the agent paradigm right from analysis; (ii) it should naturally lead to the conclusion of whether or not it is feasible to develop the system as a MAS; (iii) it should systematically identify the components of a MAS, if the problem specifications call for an agent society; (iv) it should naturally lead to this organizational model; (v) it should produce reusable agents; and, (vi) it should be easy to apply and not require too much knowledge of agent technology.

The SONIA (Set of mOdels for a Natural Identification of Agents) methodology [8] basically embraces the previous approaches: the generation of a multi-agent architecture to solve a problem (whose conceptualization is not conditioned by the agent paradigm) and the systemization and automation of the activities of identifying MAS components. Likewise, the methodology defines an agent society model that flexibly and dynamically facilitates problem solving and can be used to integrate indispensable legacy systems.

The phases and stages of which the SONIA methodology is composed are listed below, along with the models generated in each stage (Fig. 2):

– *Conceptualization*: The problem is analyzed using the Set Theory Based Conceptual Model (SETCM) [9,10], an analysis method that was defined to combine a formal foundation with a pragmatic approach. This analysis method is design-independent.

The result is an *Initial Structural Model*, which describes the overall structure of the domain (concepts, associations, attributes, classifications, etc.) and an *Initial Task Model*, which describes the problems to be solved (tasks) and the task decomposition and control of the resolution of the subtasks (task methods).

– *Extended Analysis*: Having conceptualized the problem, the models built are refined and expanded to capture the system environment and external entities.

  The Extended Analysis Stage produces the following models: an *Environment Model*, which defines the external system entities and system interactions with these entities; a *Structural Model*, which can extend the system knowledge with knowledge that the external entities supply to the system; and a *Task Model*, which can extend the tasks performed by the system with any tasks required to interact with external entities.

– *Synthesis*: This stage provides the building blocks for the component-driven bottom-up identification of agents that is performed during the design. In this process, basic elements are identified first (bottom level) and are used to identify and define agents (top level).

  It produces the following models: a *Knowledge Model*, which identifies the knowledge blocks inherent to the problem by grouping concepts and associations from the Structural Model; a *Behavior Model*, produced by grouping tasks, subtasks and methods from the Task Model; a *Responsibility Model*, output by establishing the relationships between knowledge blocks and behaviors, and a *Goal Model*, which represents the main objectives of the system. This stage bridges domain-dependent analysis and (software) solution-dependent design and is described in detail under point 3.

– *Architectural Design*: The purpose of the second stage of Multiagent Architecture Design is to define the architectural elements by means of the following models: an *Agent Model*, which identifies and defines, from the Knowledge, Behavior, Responsibility and Goal Models, what elements should be designed as autonomous agents; an *Object Model*, which identifies and defines, from the Knowledge and Responsibility Models, what passive elements there are in the environment; and an *Interaction Model*, which identifies and defines the relationships between the agents and between agents and objects.

  Not until the Agent Model is built is a decision made as to whether the architecture can be implemented by means of agents or a different paradigm needs to be used. This choice is chiefly based on whether or not agents can be identified. For an entity to be able to considered as an autonomous agent, it should have a behavior and the right knowledge blocks to perform the tasks of this behavior, have at least one defined goal and one utility, and perceive and act in the environment.

  If no agents can be identified, another design paradigm will have to be chosen. One possibility would be an object-oriented design, reusing Object and Interaction Models. Another possibility would be to design the system as a knowledge-based system, reusing the Knowledge, Behavior and Responsibility Model.

– *Society Design*: The Multiagent Architecture Design can result in an *agent society*, in which the system is designed as a set of agents embedded in a social structure. Several models are generated in this stage: a Social Agent, a Role, a Relationship, and a Social Commitment Policy [11].

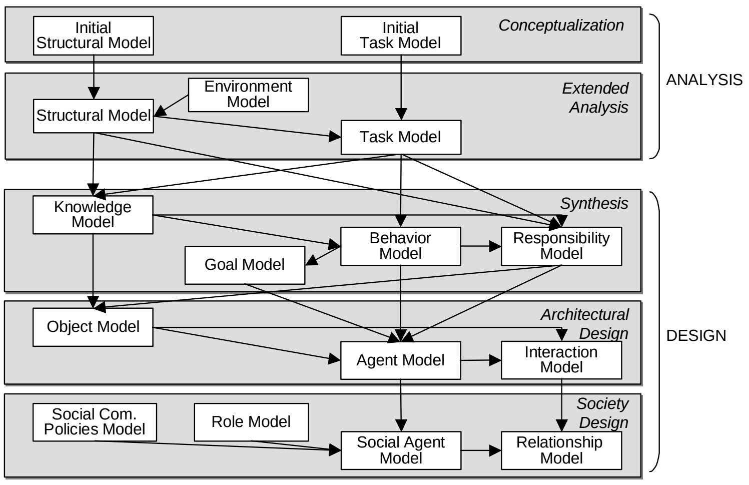


Fig. 2. Phases of the SONIA methodology

3. Synthesis Stage

As mentioned earlier, we consider a design paradigm-independent analysis to be best. In this case, the step from analysis models to architectural design models is usually a traumatic process because they are too far apart. Thus, the transition to design has to be undertaken with special care (Fig. 3).

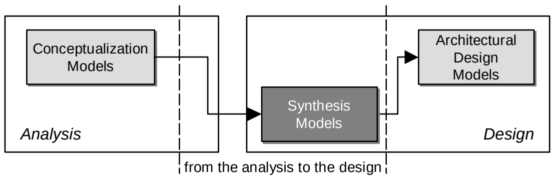


Fig. 3. Transition from analysis models to design models

This approach is based on synthesizing information of the analysis models as higher-level structures related to the reference paradigm, that is, restructuring the analysis information to adapt it to the design tools. Synthesis is the first design phase, whereby the viewpoint switches from the domain (analysis) to the solution (design). Consequently, it should ease the identification and formalization of computable structures that are coherent with agent orientation from the analysis models.

The Synthesis stage produces the models that are used later for the component-driven identification of agents. For this purpose, it re-groups elements of the Structural and Task Models to produce four other models: a Knowledge Model, a Behavior Model, a Responsibility Model, and a Goal Model.

The process of outputting these models and their application to a real case study, ALBOR (Barrier-Free Computer Access)[1], is described in the following sections [10]. ALBOR was conceived as an Internet-based intelligent system designed to provide guidance on the evaluation of disabled people's computer access skills and on the choice of the best suited assistive technologies.

3.1 Knowledge Model

The *Knowledge Model* can identify the knowledge blocks by grouping Structural Model concepts and associations. The knowledge blocks will be used internally or shared by the agents.

These clusters are identified on the basis of the concepts and associations of which they are composed, which meet the following conditions:

– They are strongly related to each other. The clusters are internally highly cohesive.
– They have little relationship to the other concepts and associations (low-coupled grouping).
– They are used to perform the same tasks.

The activities to be performed to output the first version of the Knowledge Model are:

1. Identify clusters of concepts and/or associations
2. Identify relations between knowledge blocks
3. Describe knowledge blocks

A new technique modifying Kelly's Trait Analysis [12] was used to identify concepts and associations. This technique identifies clusters of concepts and associations that have the first two properties of a knowledge block: high cohesion and low coupling.

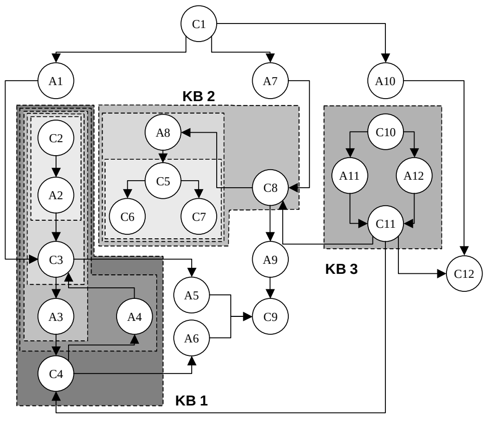
To systematize the application of this technique, the conceptual diagram of the Structural Model is first transformed into a directed graph, as follows:

– For each concept and association create a graph node.
– For each association, create an arc from each association source concept node to the association node and an arc from the association node to the association target concept node.
– For each classification, create an arc from each subconcepts node to the superconcept node.

Then a 2D table is built that stores the connectivity (number of arcs or connections) between each pair of nodes in the graph. Finally, the Trait Analysis-based technique is applied using the connectivity between two graph clusters as a measure of comparison. The technique will involve iteratively applying the clustering algorithm until no more concepts and/or associations can be clustered.

Fig. 4 shows the conversion of the ALBOR conceptual diagram output during analysis into a directed graph and groupings of concepts/associations (knowledge blocks). In this step, we obtain three knowledge blocks *KB1*, *KB2* and *KB3*.

[1] ALBOR is a project funded by the UPM (Technical University of Madrid) and IMSERSO (Spanish Institute of Migrations and Social Services).



CONCEPTS	ASSOCIATIONS
C1: User	A1: HasResponses
C2: Section	A2: HasQuestions
C3: Question	A3: HasResponses
C4: Response	A4: NextQuestion
C5: WebSite	A5: HasMedium
C6: WebApplication	A6: HasMedium
C7: WebPage	A7: HasRecommendations
C8: Recommendation	A8: HasLinks
C9: Medium	A9: HasMedium
C10: Rule	A10: HasAptitudes
C11: Fact	A11: HasAntecedents
C12: Aptitude	A12: HasConsequent

Fig. 4. Clusters of concepts/associations (knowledge blocks)

This technique makes no distinction between concepts and associations when transforming the conceptual diagram into a graph. Therefore, we will have to apply the rules listed in Table 1 to check whether or not the clusters are valid.

Table 1. Rules for dealing with invalid relations (*r*) between knowledge blocks (*k*) / concepts (*c*) / associations (*a*), siendo *s* un concepto o una asociación perteneciente al origen de *a*

Components	Rules for dealing with invalid relations
r (*k*, *c*)	R1: IsAssociation(*r*) $\wedge$ IsSource(*c*, *r*) $\rightarrow$ Add *c* to *k*
	R2: IsAssociation(*r*) $\wedge$ IsTarget(*c*, *r*) $\rightarrow$ Do not add *c* to *k* {*r* will be dealt with at later stages}
	R3: IsClassification(*r*) $\wedge$ IsSubconcept(*c*, *r*) $\rightarrow$ Add *c* to *k* {*c* belongs to knowledge block containing the classification's superconcept}
	R4: IsClassification(*r*) $\wedge$ IsSuperconcept(*c*, *r*) $\rightarrow$ Do not add *c* to *k* {*r* will be dealt with at later stages}
r (*k*, *a*)	R5: $\neg$HasAttributes(*a*) $\wedge$ Source(*a*) = {*s*} $\wedge$ (Cardinality(*s*, *a*) = 0..1 $\vee$ Cardinality(*s*, *a*) = 1..1) $\wedge$ $s \in k$ $\rightarrow$ Add *a* to *k*
	R6: $\neg$HasAttributes(*a*) $\wedge$ $\forall s \in$ Source(*a*) (Cardinality(*s*, *a*) = 0..* $\vee$ Cardinality(*s*,*a*) = 1..*) $\wedge$ $\forall s \in$Source(*a*) $s \in k$ $\rightarrow$ Add *a* to *k*
	R7: HasAttributes(*a*) $\vee$ ($\exists$ $s \in$Source(*a*) (Cardinality(*s*, *a*) = 0..* $\vee$ Cardinality(*s*, *a*) = 1..*)) $\vee$ ($\exists k_1, k_2$ $\exists s_1, s_2 \in$Source(*a*) $k_1 \neq k_2 \wedge s_1 \in k_1 \wedge s_2 \in k_2$) $\rightarrow$ Do not add *a* to *k* {*r* will be dealt with at later stages}

The knowledge blocks identified in Fig. 4 are modified as a result of applying these rules. Associations *A5* and *A6* are added to the knowledge block *KB1* applying rule *R5*. *C12* is added to *KB3* applying *R3* and *A9* is added to *KB2* applying *R5*.

The resulting clusters of the first version of the model can only satisfy the conditions of being highly cohesive and fairly unrelated to other clusters. The final version of the model, which is output when the Responsibility Model is completed, will meet all the conditions.

3.2 Behavior Model

The *Behavior Model* is the result of grouping tasks and methods of the Task Model. The behaviors will be part of the agents.

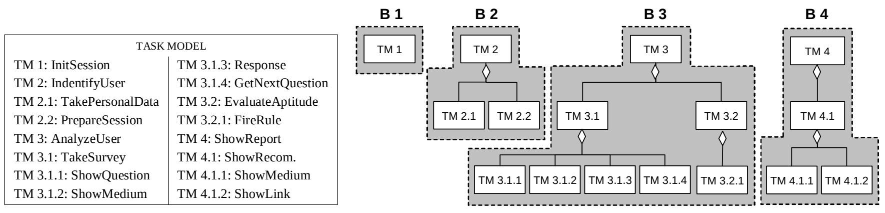
These clusters are characterized because the tasks and subtasks of the grouping:

– Depend on others through task methods.
– Use the same knowledge blocks for problem solving.

The activities to be performed to output the first version of the Behavior Model are:

1. Identify clusters of tasks and/or subtasks
2. Identify time dependences between behaviors
3. Describe behaviors

In this first version of the Behavior Model, one cluster is generated for each first-level task in the task/method diagram of the Task Model, adding its subtasks to the cluster. Fig. 5 shows the cluster of ALBOR tasks/methods (*B1*, *B2*, *B3* and *B4*). The time dependences between behaviors are then calculated from the preconditions and postconditions of these clusters. A behavior *B2* depends on a behavior *B1*, if there is a knowledge block (and concepts and/or associations) in the postcondition of *B1* that is also in the precondition of *B2*.


TASK MODEL

TM 1: InitSession	TM 3.1.3: Response
TM 2: IndentifyUser	TM 3.1.4: GetNextQuestion
TM 2.1: TakePersonalData	TM 3.2: EvaluateAptitude
TM 2.2: PrepareSession	TM 3.2.1: FireRule
TM 3: AnalyzeUser	TM 4: ShowReport
TM 3.1: TakeSurvey	TM 4.1: ShowRecom.
TM 3.1.1: ShowQuestion	TM 4.1.1: ShowMedium
TM 3.1.2: ShowMedium	TM 4.1.2: ShowLink

Fig. 5. Clusters of tasks/methods (behaviors)

The resulting clusters for the first version of the model can only satisfy the condition of interdependency through task methods. The final version, which is output when the Responsibility Model is completed, will meet all the conditions.

3.3 Responsibility Model

The *Responsibility Model* is output by relating knowledge blocks to behaviors. This model is essential for identifying agents and environment objects. A basic activity is to refine the Knowledge and Behavior Models to meet all their conditions.

The activities to be performed to output the Responsibility Model and the final versions of the Knowledge and Behavior Models are:

1. Identify the use relations of concepts/associations/knowledge blocks in behaviors
2. Modify the Knowledge Model
3. Modify the Behavior Model
4. Identify responsibilities between knowledge blocks and behaviors
5. Describe responsibilities

The Knowledge and Behavior Models are refined first by identifying the use relations between concepts/associations/knowledge blocks in behaviors and incorporating in a 2D table. A mark will be placed at the intersection between a behavior with a concept/association/knowledge block if they are used to perform a task/subtask belonged to that behavior (that is, if it appears in the precondition or postcondition of the task/subtask) (see Table 2).

Table 2. Relations of use between concepts/associations/knowledge blocks and behaviors

		Behaviors			
		B1	B2	B3	B4
Concepts / Aassociations	C1	X	X	X	X
	A1			X	
	A7			X	
	C9			X	X
	A10			X	
KnowledgeBlocks	KB1			X	
	KB2				X
	KB3			X	

To output the final Knowledge Model, all the concepts/associations will have to be included in one knowledge block. The rules in Table 3 can be used to determine when to add a concept/association to an existing knowledge block or when to cluster a number of concepts/associations as a new knowledge block. These rules of inclusion will be based on the use relations of concepts/associations in output tasks/subtasks.

Table 3. Rules of inclusion of concepts (*c*) / associations (*a*) of the unclustered Structural Model (*usm*) in knowledge blocks based on relations *X*:*Y* (where *X* are tasks/subtasks and *Y* are concepts/associations)

Inclusion Rules
(*r* = 1:1 ∨ *r* = 1:N) ∧ (*c*/*a* are not related to other elements of *usm*) ∨ (c/a are related to other elements of *usm* ∧ the constraints in Table 1 hold) → Cluster *c*/*a* in an existing knowledge block
(*r* = *N*:*M*) ∧ (*c*/*a* are not related to other elements of *usm* ∨ (*c*/*a* are related to other elements of *usm* ∧ the constraints in Table 1 hold) → Cluster *c*/*a*∈*r* to form a new knowledge block

The following actions are taken taking into account the relations of use (Table 2) and inclusion rules (Table 3). Concepts *C1* and *C9* have a N:M relation with more than one behavior, which means that two knowledge blocks (*KB4* that contains the concept *C1* and *KB5* the concept *C9*) are created. On the other hand, concepts *A1*, *A7* and *A10* have a 1:N relations with behavior *B3* and can be added to the knowledge to which they are related (*A1*, *A7* and *A10* are included in *KB4*).

To output the final Behavior Model, the behaviors will have to meet the second of the conditions set out in its definition. This condition demands that the same knowledge block should be used to perform the tasks of a single behavior. The rules in Table 4 can be used to evaluate whether more than one behavior should be clustered as one or divided into more than one behavior on the basis of the use relations of concepts/associations in the output tasks/subtasks.

Table 4. Rules of behavior clustering/division based on relations (*r*) *X*:*Y* (where *X* are behaviors and *Y* is knowledge block)

Clustering/division rules
r = 1:*N* → divide behaviors into more than one behavior. {Algorithm: each behavior will contain a subtask of the next level of decomposition of the main task of the divided behavior. Analyze the relations between the new behaviors and the new knowledge block to check that they are 1:1. If so, stop; otherwise, divide the behaviors again}
r = *N*:1 → cluster the behaviors into one

As a result of applying these rules, behavior *B3* is divided into two behaviors in ALBOR. On the other hand, there were no behavior clusters.

3.4 Goal Model

The *Goal Model* is composed of the system objectives. The aim of this model is to be able to identify agent goals. The agent will execute behaviors to achieve its goals.

The activities to be performed to output the Goal Model are:

1. Identify goals
2. Describe goals

These goals are logical conditions imposed on the state of knowledge and are identified from Behavior Model task postconditions. The goals are a subset of the union of the postconditions of the behaviors that define the system goals. The designer's task is to decide which subset of postconditions defines the system goals.

In ALBOR, a single goal "new Users.recommendations" has been identified for the EvaluateAptitude behavior (B4), whose meaning is to get new recommendations for a user.

3.5 Transition to Architectural Design phase

The agents and objects were identified during architectural design from the Responsibility Model. The knowledge shared by several behaviors was chosen as *objects*. Following this criterion, we identified the "Users" and "Media" objects (white box in Fig. 6).

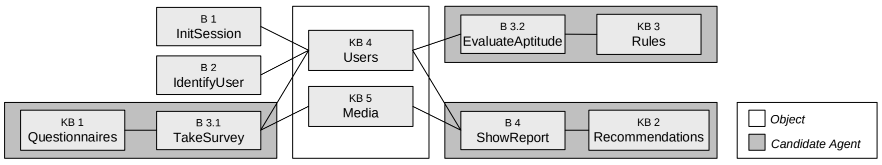


Fig. 6. Identification of objects and candidate agents

A *candidate agent* will be output for each knowledge block that is the responsibility of a sole behavior (relation 1:1). In the case of ALBOR, three candidate agents were identified: 'Survey-Taker' agent (*B3.1* and *KB1*), 'Decision-Maker' agent (*B3.2* and *KB3*) and 'Advisor' agent (*B4* and *KB2*) (grey box in Fig. 6). The *B1* and *B2* were not assigned, because they do not have any proper knowledge.

Not all candidate agents will be converted to agents. This will be confined to agents that meet all the requirements for becoming an *autonomous agent*, i.e. agents that have at least one defined goal and utility, and sense and act in the environment. In ALBOR, the three candidate agents identified earlier meet these two conditions and qualify as autonomous agents.

4. Conclusions

This paper aims to contribute to the methodological issue of agent-based development by defining a new methodology, SONIA, that includes a synthesis stage to smooth the transition between the design paradigm-independent analysis and paradigm-dependent multiagent architectural design. For this purpose, we described the models underlying this synthesis stage (knowledge, behavior, responsibiliy and goal model) and the mechanisms for building them.

References

1. Blum, B. I. Beyond Programming, Oxford University Press, New York (1996)
2. Shapiro, S. Splitting the difference: the historical necessity of synthesis in software engineering. IEEE Annals of the History of Computing, 19(1) (1997) 20-54
3. Alonso, F., de Antonio, A., González, A. L., Fuertes, J. L., Martínez, L.A.: Towards a Unified Methodology for Software Engineering and Knowledge Engineering. Proc. of the IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC'98), San Diego, USA (1998) 4890-4895
4. Bresciani, P, Giorgini, P, Giunchiglia, F and Mylopoulos, J Tropos: An Agent Oriented Software Development Methodology, Int. J. of Autonomous Agent and MultiAgent System, 8(3) (2004) 203-236
5. Zambonelli, F, Jennings, N R, and Wooldridge, M Developing Multiagent Systems: The Gaia Methodology, ACM Transactions on Software Engineering and Methodology, 12(3) (2003) 317-370
6. Padgham, L and Winikoff, M Prometheus: A Methodology for Developing Intelligent Agents, in: Giunchiglia, F, Odell, J, and Weiss, G (eds.): Agent-Oriented Software Engineering III, LNCS 2585, Springer-Verlag, Heidelberg (2003) 174-185
7. Iglesias, C A, Garijo, M, González, J C, and Velasco, J R Analysis and Design of Multiagent Systems using MAS-CommonKADS, in: Singh, M P, Rao A S, and Wooldridge, M (eds.): Intelligent Agents IV: Agent Theories, Architectures, and Languages (ATAL97), LNAI 1365, Springer-Verlag, Heidelberg (1999) 313-326
8. Alonso, F., Frutos, S., Martínez, L.A., Montes, C.: SONIA: A Methodology for Natural Agent Development. In: Gleizes, M. P., Omicini, A., Zambonelli, F. (eds.): Engineering Societies in the Agents World V, LNCS/LNAI 3451, Springer-Verlag, Heidelberg (2005) 245-260
9. Martínez, L.A.: Method for Independent Problem Analysis. PhD Thesis. Universidad Politécnica de Madrid. Spain (2003)
10. Alonso, F., Frutos, S., Fuertes, J. L., Martínez, L. A., Montes, C.: ALBOR. An Internet-Based Advisory KBS with a Multi-Agent Architecture. Int. Conference on Advances in Infrastructure for Electronic Business, Science, And Education on the Internet (SSGRR 2001), L'Aquila, Italy (2001) 1-6
11. Alonso, F., Fernández, F., López, G., Rojas, F., Soriano, J.: Intelligent Virtual Agent Societies on the Internet. In: de Antonio, A., Aylett, R., Ballin, D. (eds.): Intelligent Virtual Agents, LNCS/LNAI 2190, Springer Verlag, Heidelberg, Berlin (2001) 100-111
12. Kelly, G. A.: The Psychology of Personal Constructs. Norton (1995)