# Multi Agent System Development Kit

Vladimir Gorodetsky, Oleg Karsaev, Vladimir Samoylov,
Victor Konushy, Evgeny Mankov and Alexey Malyshev

**Abstract.** Recent advances in the area of multi–agent technology are attracting a growing attention and interest of both scientific community and industry. This interest is stipulated, on the one hand, by the steadily increasing capabilities of multi-agent technology that offers a new paradigm and powerful means for design of large scale distributed intelligent systems, and, on the other hand, by the practical needs of industry to have a reliable and efficient technology to cope with new challenges of practice. At present, one of the most important research challenges is elaboration of powerful methodologies for agent-based systems engineering and development of efficient software tools supporting implementation and deployment of the multi-agent systems. The paper presents one of such tools, Multi–Agent System Development Kit, based on and implementing *Gaia* methodology that supports the complete life cycle of multi-agent system engineering, implementation and deployment, and insures the integrity of all the solutions produced by designers at different stages of the development process.

## 1 Introduction

Although agent-oriented software engineering has been a subject of intensive research for over a decade, it has not yet reached the level of maturity required for it to be rated as an industrial technology. By now a lot of MAS software tools have been developed. Among them, the well-known and highly popular are AgentBuilder [20], Jack [18], JADE [3], ZEUS [9], FIPA-OS [13], agentTool [11], etc. An almost complete list of such software tools can be found in ([24], [25]). Nevertheless, in spite of the rich theoretical achievements in the area, there practically exist no Multi-Agent System (MAS) software tools capable of supporting the complete life cycle of industrial MAS comprising analysis, design, implementation, deployment and maintenance.

A capabilities study of the existing software tools allows to see a number of common disadvantages considerably decreasing their performance and degree of maturity. The first of such disadvantages to be mentioned, and a substantial one, consists in insufficient usage of the experience accumulated within the object–oriented approach and the existing (at least, de-facto) "standards" developed for the analysis, design and implementation stages that are commonly used in the information technologies [6]. The major source of it is the fundamental mismatch between the methods and the abstract conceptions used by the object-oriented approach [6], on the one hand, and by the agent-oriented approach ([22, 23]) on the other hand, explicitly shown in [21]. A number of initiatives is being currently undertaken to overcome this mismatch and to find an efficient solution. Among them, the most promising one is the initiative being jointly undertaken by two leading international organizations focused on standardization in the area of new information technologies, FIPA and OMG, as a part of project *Agent UML* [1].

Another considerable disadvantage of the existing MAS software tools is a weak consistency of the solutions produced at the consecutive stages of a MAS application life cycle. As for this, a certain growth of research activity can be observed in the development of methodologies aimed at supporting the above mentioned consistency of the MAS life cycle, including the analysis, design, implementation, deployment and maintenance aspects. Some well-known methodologies of such kind are Gaia [21], MESSAGE [7], MaSE [12], Prometheus [19], Adelfe [4], Tropos [14], PASSI [8]. The core of the integrity maintenance problem is the necessity to automatically maintain the consistency of the solutions being produced at the different stages of the MAS development process. For example, while implementing MAS, it is necessary to strictly conform to the solutions produced at the analysis and design stages. Rational Rose software tool supporting an information system object-oriented analysis, design and implementation, can serve as a guiding line for this purpose. Comparison of MAS development tools and methodologies can be found in ([5], [10]).

This paper presents the Multi-Agent System Development Kit (MASDK) that was a subject of research and development during the last four years. The first and second versions of MASDK [15] were practically used for fast prototyping of several MAS applications in different problem domains [16, 17]. The experience accumulated during this period as well as recent advances and trends in MAS methodology and technology made clear the fundamental drawbacks of the earlier versions and gave way to forming sound requirements for the next version of MASDK, version 3.0.

For this version design some new trends and achievements in the agent-oriented software engineering methodology have been taken into account. In particular, the basic methodological principles of the Gaia methodology [21] have been implemented. Indeed, Gaia methodology consists of two basic stages of MAS development that are (1) analysis and (2) design. The analysis is aimed at reaching *"an understanding of the system and its structure (without reference to any implementation detail)"* [21]. This stage assumes development of abstract solutions and notions related to the system organization, i.e. identification of MAS tasks, description of role models, and interaction model [21]. An objective of the design stage is *"to transform the abstract models derived during the analysis stage into models at a sufficiently low level of* *abstraction that they can be easily implemented"* [21]. This stage assumes formal specification of agent models, service models, and acquaintance model.

MASDK 3.0 software tool is currently being assessed and evaluated by means of developing of a number of multi-agent systems, e.g., for intrusions detection in computer network, for situation assessment and for business activity monitoring.

This paper describes the MASDK 3.0 software tool, and with a certain focus on its correlation with the Gaia methodology. Section 2 outlines MASDK 3.0 software tool, technology supported by it, and abstract agent architecture. Section 3 covers the overall view of the specification structure of multi-agent systems developed using MASDK, outlines development methodology and its mapping with Gaia one. The analysis and design stages, supported by MASDK and carried out through usage of a number of specialized graphical editors are considered in section 4. Section 5 outlines the implementation and deployment stages. Section 6 considers one of the most important problems of any MAS technology which consists in maintaining consistency of solutions produced at various technology stages; it is also shown in the section how the consistency is insured within a technology supported by MASDK. Related works and comparison of MASDK with some other tools is presented in Section 7. In conclusion the paper basic results are summarized.

## 2 Outline of MASDK 3.0 Software Tool and Technology Supported

MASDK 3.0 software tool consists of the following components (Fig.1): (1) System core which is a data structure designed for XML-based representation and storing of MAS formal specification; (2) Integrated set of graphical editors supporting the user's activity aimed at formal specifying of the MAS under development; (3) Library of C++ classes, comprising what is usually called Generic agent, corresponding to the reusable

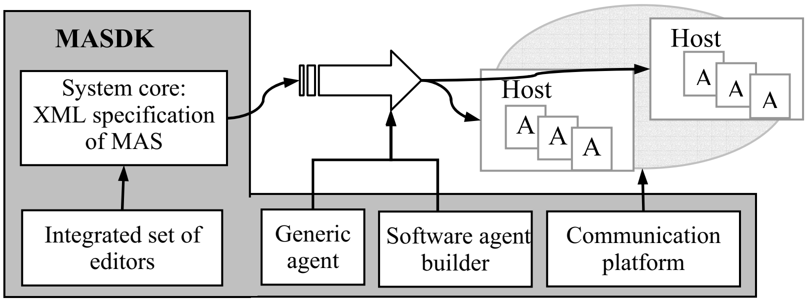


**Fig. 1.** MASDK software tool components and their interaction

component common for all agents; (4) Communication platform to be installed in the computers, in the network, that are involved in the MAS being designed, and (5) Builder of software agent instances, that carries out (i) generation of source and executable codes of software agents in C++ language and (ii) deployment of software agents over the earlier installed communication platform.

Generalized architecture of the agents developed using MASDK is shown in Fig.2. It includes the following basic components:

- invariant (reusable) component called *Generic Agent*,
- agent behavior model,
- mental model of agent,
- set of agent services represented in terms of state machines, and
- library of auxiliary application-oriented functions.

The *Generic agent* component which is the basic class for all agents is a reusable component of MASDK environment (Fig.1). This component can be thought of as *engine* that realizes a common (reusable) scenario of agent operation. It comprises the general functions that 1) support sending and receiving of messages, 2) initiate respective services of agents, control of their execution and process of interruptions depending on the current states of agents, 3) provide the interaction agents with human users, 4) provide access to agents' data storages, etc.

*The Agent behavior model*, *Mental model* and *State machines* are intended for problem-oriented specification of agent. The agent behavior model specifies the states associated with providing the agent with respective services, the models of services being specified as state machines. The mental models comprise specification of notions, data storage structure, and the initial data and knowledge possessed by the agent. All three mentioned classes of components are developed by a designer using the *integrated set of editors* (Fig.1) and stored in *system core* (Fig.1) as *XML*-based specifications.

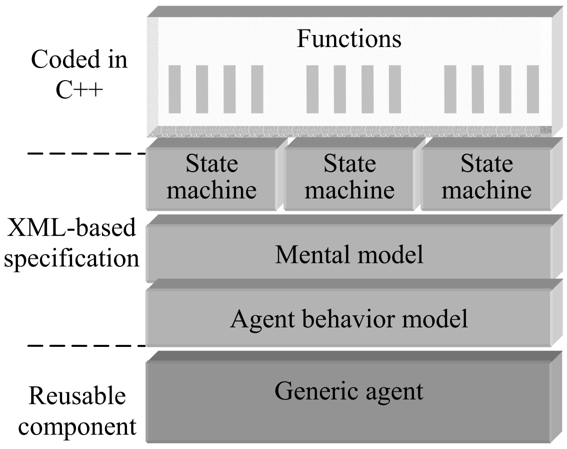


**Fig. 2.** Agent architecture

The *Functions* specify agent operation in the states of state machines. They are coded in the usual way, i.e., in MASDK in C++ language, and can be of one of the two varieties: *scripts* and *external functional components*. Scripts are fragments of software code that are developed using corresponding MASDK editors and stored in the system core as entries of MAS XML-based specification. External functional components are developed out of the MASDK environment and viewed as components the execution of which is initiated by the agent during execution of scripts.

A specification of MAS in MASDK includes specifications of *agents* and *agent classes*. Agents of each class have a common specification of the components comprising the architecture of agent (Fig.2), except for the content of mental models, in particular – the initial data and the knowledge possessed by the agents.

Software implementation of agents is performed by the *Software agent builder* component (Fig.1) in automatic mode. For each agent class, the software agent builder executes the three successive tasks:

1. Generation of the source C++ code of the problem–oriented components of agent class, i.e. of the *Agent behaviour model* and all *State machines.* The main goal of this task is transformation of *XML*-based specifications of these components into source code in C++ on the basis of *XSLT* technology.
2. Compilation of the C++ code of agent class into an executable code. For this procedure the following components are used as inputs:
   - Source C++ code of the problem–oriented components produced by the task 1;
   - Source C++ codes of particular functions;
   - *Generic agent* components, which are reusable components of MASDK.

   In case of an error, the generator analyses the compilation output file and localizes the state machine, also indicating the state which contains the error. If a syntax error is made by a designer who was responsible for specifying the algorithm of the agent behavior, the generator indicates the name of the state machine, its state and the number of the row, in which the designer made the error.
3. Building the data storage (storages) specified in mental models of the agent class.

## 3 Methodology of MAS development

A MAS development process in MASDK can be technologically divided in to three stages. At the first stage a detailed design of the MAS is developed. In particular, this stage assumes the development of the models described in Gaia methodology [20], namely – role models, interaction model, agent models, service models, and acquaintance model. At that the service models of agent classes are described in greater detail than it is done with Gaia methodology. All activities at this stage are entirely carried out by the *integrated set of editors (ISE)* and the results (the detailed design of the MAS) are stored at the *system core* component in special XML-based language. The second stage consists in programming the particular components that can be derived from the detailed design of the MAS developed at the previous stage. The third stage consists in compiling and building the software agents.

Thus, almost all development process is carried out based on the *ISE* component; so, let us discuss this component and the respective methodology of application systems development in detail. The set of editors comprising the *ISE* component can be divided into three categories (Fig.3): 1) the set of editors (*MAS model, Protocol, Ontology* editors) aimed at describing abstract concepts and system organization, 2) the set of editors (*Agent class behavior model, State machine, State, Private ontology* editors) designed for description of concrete concepts and design of agent classes, and 3) the set of editors (*Agents' configuration and Agent mental model* editors) aimed at building and deployment of the agents.

The enumerated set of editors provides the following methodology for MASs development that starts from the two initial stages (*analysis* and *design*) which are parts of Gaia methodology [21].

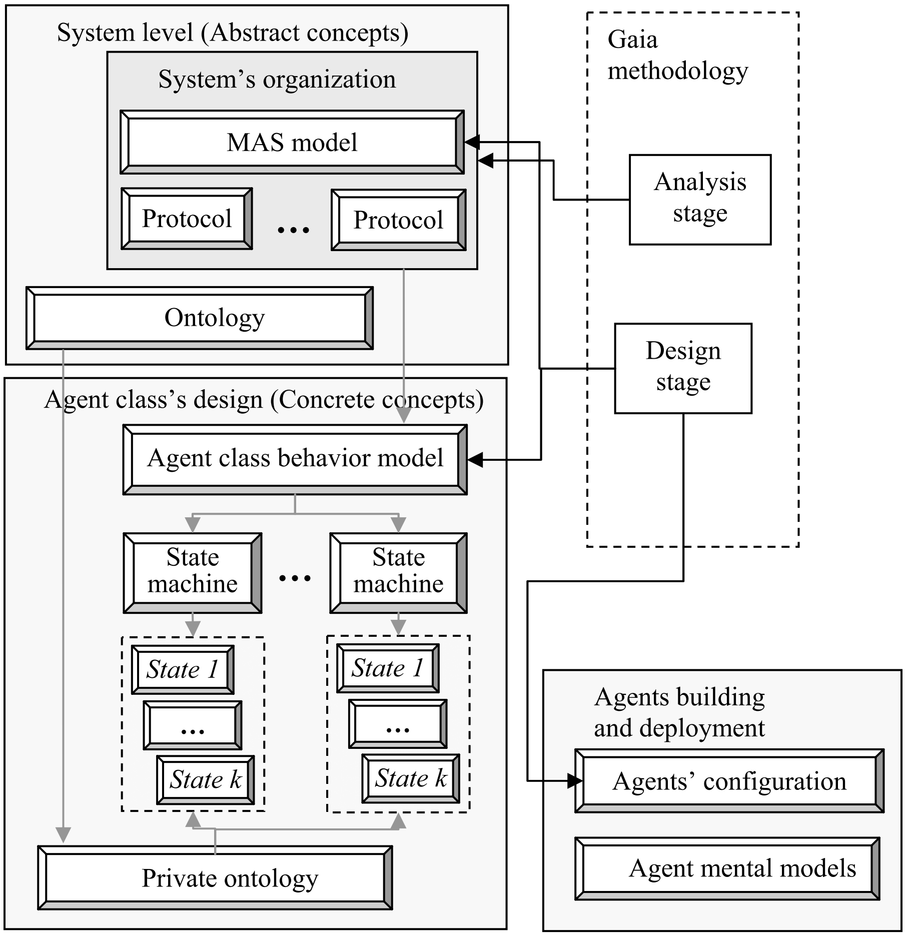


**Fig. 3.** Structure of graphic editors in MASDK and their mapping with Gaia methodology

1) *Analysis.* The development process is initiated with the requirements statement and the results of *role* and *interaction* models specification that comprise *an organizational model* of the MAS. The respective activities are supported by the *MAS mode*l and the *Protocol* editors.

2) *Design.* According to Gaia methodology the objective of this stage assumes development of *agent, service* and *acquaintance* models. The agent model development assuming determination of agent classes and instances of each from them is carried out using the *MAS model* and the *Agents' configuration* editors. The service models are

developed using the *Agent class behavior model* editor and the acquaintance model with the aid of the *MAS model* one.

3) *Ontology description*. Domain ontology description is specified by means of the *Ontology* editor and initially[1] can be developed in parallel with stages 1 and 2. However there are relations between the models presupposes a certain sequence of their development. In particular, a detailed specification of communication acts of the protocols at later phases of development implies specifying of messages contents in terms of ontology notions. It means the respective notions of ontology have to be specified for doing it. The first three stages are designed for describing abstract concepts of the application system. The concrete concepts are developed through the following sequence of stages.

4) *State machine development (Service development)*. Each service identified at the design stage (stage 2) is developed as a state machine using *State machine* and *State* editors. At that, such a state machine is described at the level of states and transitions between them without minor details of states implementation. A service is associated with respective protocols and activities and, consequently, development of a state machine assumes relating each state to either communication act of some protocol or one of the activities. One communication act is related to only one state, while one activity can be related to several states. Transitions between states describe the scenario of the service execution.

5) *Private ontology development*. Private ontology is required for specification of the notions used for describing the agent class mental model. At that, it involves private notions of the agent class and the notions inheriting ones from the shared ontology developed at stage 3. Private ontology development also assumes specifying a data storage scheme (or several storage schemes) of agent class.

6) *Agents initial mental model*. This stage consists in describing initial data and knowledge of the agents. It must be noted that at the previous stages (namely – 4, 5) the agent classes have been specified and the agents of each class have no specifics except for their names. Therefore, the initial mental models of agents are their specifics that single out the agents of each class.

7) *Agent classes components programming*. Components of each agent class that have to be specified in the usual way (in case of MASDK environment it consists in developing a code in C++ language) are identified as a result of stage 3 execution. All of them are either scripts of agent classes' behavior in particular states of state machines or invoked from these scripts as external components.

8) *Agents code generating*. This function is executed automatically by *Software agent builder* component (Fig.1).

9) *Agents configuration and deployment*. This stage assumes specifying the locations of agents, deploying of the agent according to the results, and filling in the storages of the agents with their initial mental models developed at stage 6.

[1] Development process in MASDK can have iterative character. It means each model developed in process of MAS development can be refined repeatedly.

Thus a development of MAS in MASDK is generally reduced to development of mentioned problem-oriented components. It is carried out using of graphical editors through which the respective *XML*-based templates stored in system core are filled in. These graphical editors and structure of respective components are considered in the following section.

## 4 Integrated system of editors

MAS specification comprises a set of interrelated components that are developed based on respective template types and stored in the system core. Set of these components comprising MAS specification are depicted in the environment browser a screenshot of which with an abstract example is shown in Fig.4. Specification of each MAS includes the following components:

- Domain *ontology*;
- *Meta model* of MAS describing its organization, namely – names of the identified roles, agent classes, protocols and relations between them;
- Description of *role models*;
- Specification of *protocols*.

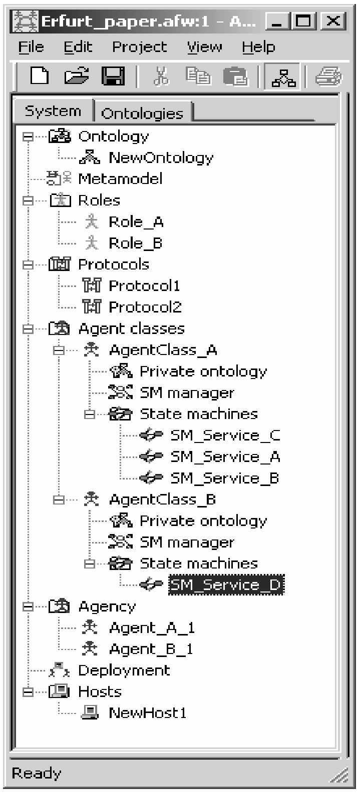


**Fig. 4.** Browser and System core content

Model of each agent class comprises the following models:

- *Private ontology* describing the notions and storages of mental model;
- *SM Manager (State Machines manager)* describing list of service names and scenarios of their execution;
- Set of *state machines*, each of them specifying respective service;
- Initial mental model of each *agent* from the class;
- *Deployment* model describing configuration of agents and their locations;
- *Host* model describing necessary data of the hosts where agents are located.

The integrated editors system consists of seven basic graphic editors corresponding to the respective classes of components constituting the specification of MAS under development. It means that the components of each class are developed using respective graphic editor (the main editor). Let us remark that according to the methodology of MAS development in MASDK described in section 3, a designer uses the same editors at all different stages of a process. Using a graphic editor implies using several auxiliary editors. These are used for specifying specific details of the components and not dealt with in detail in the paper.

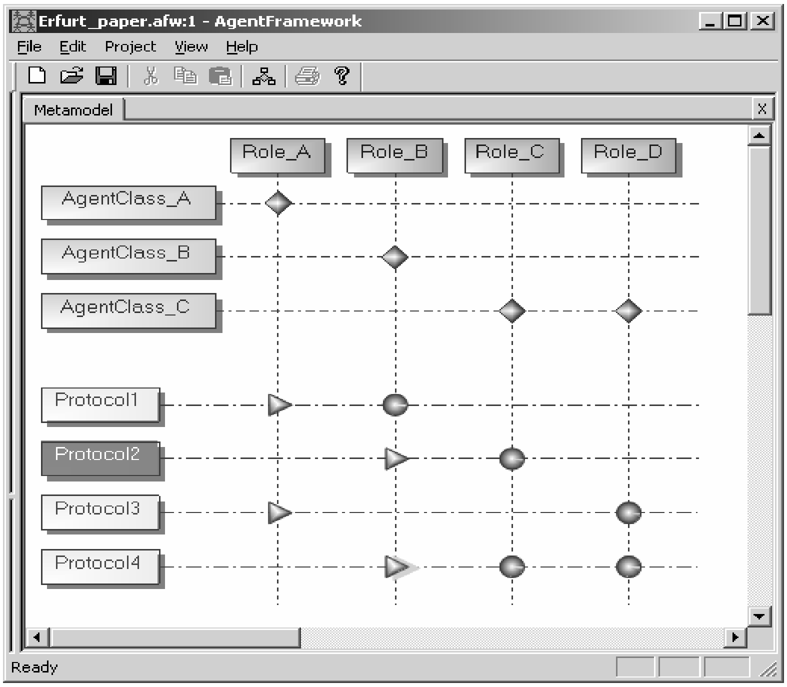


**Fig. 5.** MAS model editor

## 4.1 MAS model editor

MAS model editor (Fig.5) and related to it the auxiliary editors (dialogs) are designed for description of the MAS organization, and they also support carrying out of the two initial stages of the development process: analysis and design partially. At the analysis stage these editors are used for specifying role and interaction models. The set of roles identified/found at this stage is presented in the upper part of the model editor window. In the current version of MASDK a model of each role is specified in the auxiliary editor according to the template which is an accessory of Gaia methodology as high-level (textual) description. A formal specification of this template for graphic representation of the role models has been developed and after implementation of the respective editor it will be integrated in MASDK environment.

Description of an interaction model is implemented using two editors. The MAS model editor allows to describe a high-level representation of this model. Such a representation assumes listing of all interaction protocols, specification of all participants of each protocol, and pointing out the roles that initiate these protocols. It is presented in the bottom part of the editor window (Fig.5). The initiators of the protocols are presented as triangles. A detailed specification of each protocol is executed using another editor which is described in the section 4.2.

High-level representation of the interaction model in the model editor allows for considering additional useful tasks related to the specification of this model. In particular, one of these tasks deals with the possibility of describing the MAS behavior on the whole. This task is reduced to investigation of possible relations between protocols and using them for describing meta-scenarios of the MAS behavior. It is viewed as one of the possible improvements of MASDK environment in future. However, the current version already allows to analyze some aspects of this task. Presentation of the rectangles denoting protocols in proper order, and using different colors gives way to show protocols nesting. E.g., in the example the protocol 2 is specified as nested protocol in protocol 1.

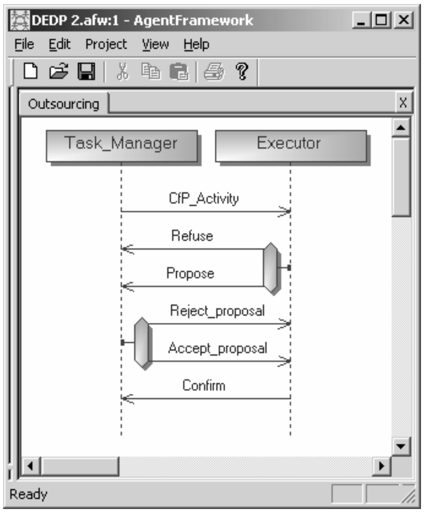


**Fig. 6.** Protocol editor

The above results, including the information about the found roles, the interaction protocols, mapping of the roles to the protocols and the textual description of the role models are used for defining the agent models, namely – the agent classes. According to the methodology this is considered to be a task of the design stage. The set of agent classes and assignment of roles to them are shown in the middle part of the MAS model editor window. Let us notice that any agent class may be able perform one or several roles. E.g., in the example, the agent class 3 is mapping two roles while two other agent classes are one-to-one mapping roles 1 and 2.

## 4.2 Protocol Editor

Detailed description of the roles interaction protocols is one of the key tasks of the analysis stage. Realizing the extremely high importance of this task was evidently one of the motives for the Agent UML project ([1], [2]) having been jointly initiated by FIPA and OMG. The project goal consists in extension of the UML language to agent-based system specification language, and one of the focuses of this Project is development of a language specially designed for specification of agent interaction protocols [2].

MASDK 3.0 includes a graphic editor of protocols. An example of a protocol graphic representation in Fig.6 within the window of editor in question is *Contract Net Protocol*. The current version of the implemented editor makes use of some basic constructs of Agent UML project. In particular, the basic capabilities of the roles interaction protocol are as follows: 1) the protocol is specified as a modified sequence diagram (Fig.6); 2) the message exchange scheme is specified using *AND*, *OR* and *XOR* connectors; 3) the participants of interaction protocols are roles; 4) the specification of message classes is implemented in ACL language. Some of the constructs being considered in Agent UML project and not included in current version of the protocol editor are several *life lines* of a role and *nested protocols*. The first construct (several life lines of a role) will be included in the next version of the protocol editor. Specification of the nested protocols in MASDK is not regarded as a task being solved using the protocol editor. The interaction model in MASDK is specified at two levels and with usage of two editors respectively. The first level of an interaction model is specified using MAS model editor (Section 4.2), and detail specification of each protocol is carried out using protocol editor. At that, specification of nested protocols is a task related to the first level with usage of MAS model protocol. This possibility is described in Section 4.1.

## 4.3 Ontology editor

A sample snapshot of the window of this editor is given in Fig.7. It aims at introducing the shared application ontology notions and the relations between them, and at specifying the private ontologies of agent classes. Description of the notions assumes introducing attributes for them and specification of the attributes domains.

The classes of relations are important characteristics of the ontology editor. At that, including them in MASDK suggests availability of respective mechanisms in the environment that have to provide for solving a number of tasks in which introduced relations are used. In particular, these mechanisms have to provide for solving the following tasks:

- Insuring correctness of the relations specifications, i.e. checking of constraints specific for particular classes of relations, e.g., introducing an inheritance between notions should prohibit cycles generation.
- Support of data storage structures generation aimed at storing the agent classes mental models.
- Providing the necessary syntactic expressiveness of the agents' mental model specification language and an object–oriented style of access to the data storages.

The existing version of MASDK includes the mechanisms that allow to implement these functions for processing inheritance relations. In the course of this processing, both single and multiple inheritance relations between the ontology notions are considered. Development and implementation of the mechanisms of this kind for other classes of relations (inclusion, association, etc) is in progress now, this task being considered as a high priority.

The inheritance relations between the classes of ontology notions described using a respective graphic editor are depicted by arrows between the notions belonging to this relation (Fig.7).

This editor is also used as the editor of agent class private ontology. Besides the abilities and functions discussed above, it is also able to perform the following functions.

Since private ontology of an agent class can include not only specific notions but some notions of the shared part of the application ontology as well, the application ontology editor has to be able to select a subset of the application ontology notions in order to use them in the private ontology of an agent class. This function is the first additional one.

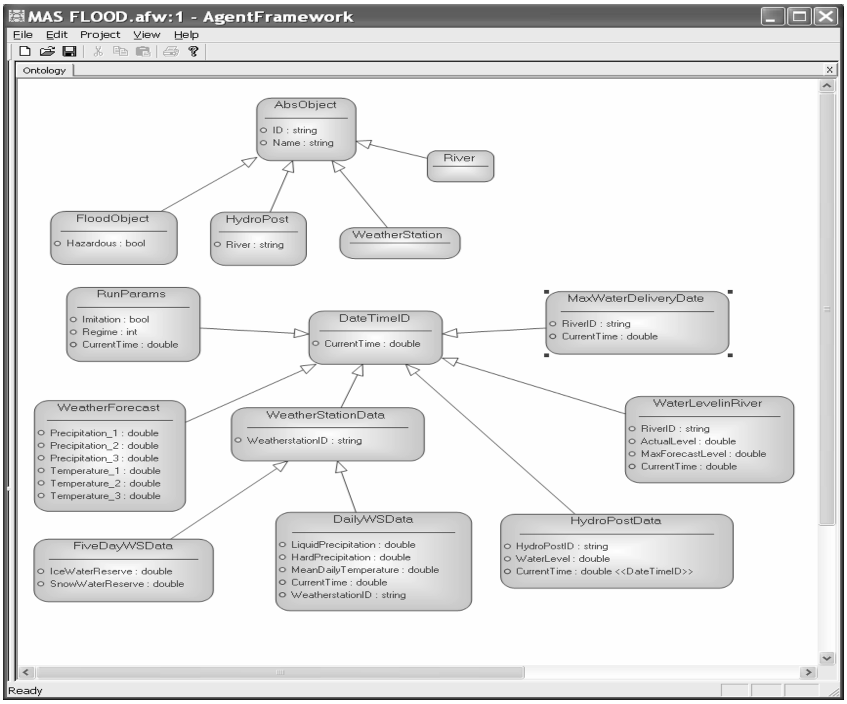


**Fig. 7.** Ontology editor

The second additional function of the application ontology editor supports specification of the storage structure for the agent class mental model. To realize this function, specification of the notions and relations of the agent class private ontology is carried out. Later, while generating agent class instances, this specification is used for automated generation of respective storages for data and knowledge. The resulting data structures can be either relational data bases or XML files.

## 4.4 Agent Class Behavior Model Editor

Agent class behavior model is designed for listing a set of agent class services and for specifying scenarios of their execution. A MAS model specification comprising the role models, the interaction model, and the assignment of roles to the agent classes form a basis for deriving a set of services for each agent class. Specification of agent class behavior model is carried out using graphic editor whose screenshot along with an abstract example of the model is shown in Fig.8. It should be noted that this editor only describes the name and conceptual description of services. More detailed formal specification of each service is carried out as a state machine model, and to do this a state machine editor described in Section 4.5 is used.

The main objective of the model consists in specifying scenarios of the derived services execution. It implies specifying the respective components defining the types of events initiating execution of services, and actions that can be or has to be performed during execution of the respective services. The model template includes the following kinds of such components.

*Input protocols*

This component lists the protocols in which the agent class under development is a participant and not an initiator. It is worth noting that this subset of protocols is derived

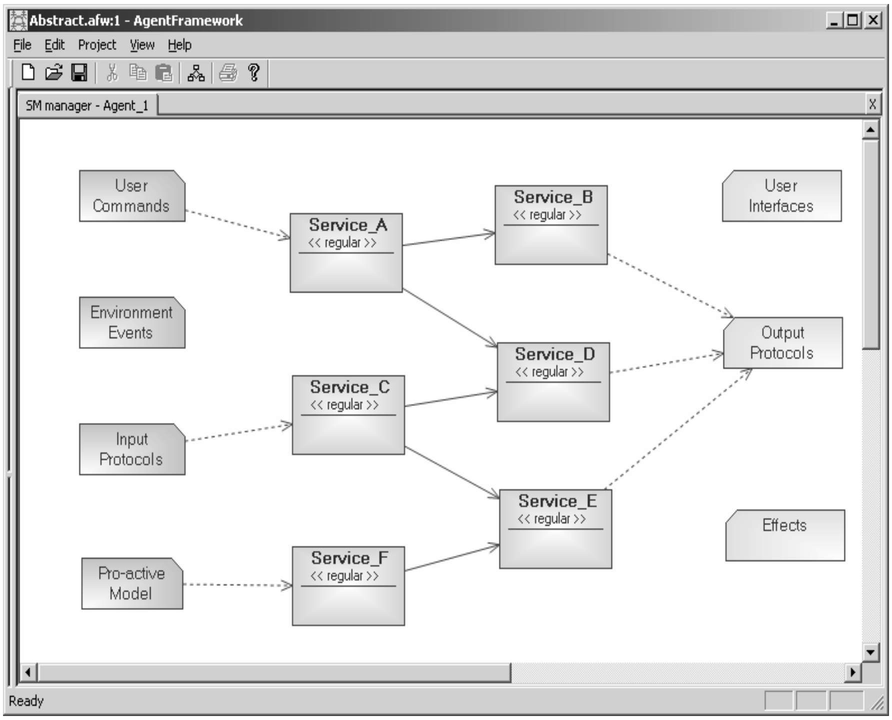


**Fig. 8.** Agent class behavior model editor

from MAS model automatically. Based on the above this component specifying is reduced to mapping of each protocol from the list to respective service. This activity is carried out through the auxiliary dialog related to this component, and in the window of agent class behavior model editor such relations are shown as arrows between the *Input protocols* component and the respective services. Each protocol from the list is one-to-one mapping respective service. At that each mapping implies the certain information for the service (state machine) development. It has to comprise states for processing of all protocol's communication acts, and the first protocol's act triggers execution of the service (state machine).

*Output protocols*

This component contains the list of the protocols in which the agent class under development is indicated as initiator. This component specifying is reduced to carrying out same activities as those being executed for *Input protocol* component specifying. This subset of protocols is automatically derived from the MAS model; each protocol from the list is one-to-one mapping respective service (state machine). This activity is carried out through the auxiliary dialog related to this component, and in the window of agent class behavior model editor such relations are shown as arrows between the respective services and the *Output protocols* component. Each mapping implies that the state machine realizing the service has to comprise states for processing of all protocol's communication acts.

*User commands*

This component is used in case when the agent class under development has interface with user, and any services are triggered by him. The component specifying is reduced to 1) listing of user commands, 2) their one-to-one mapping to respective services (state machines), and 3) identifying states of agent class when the commands are accessible. Mechanism providing for a usage of the commands is realized as a reusable function of the generic agent component. In particular, it selects accessible commands depending on the agent state and presents them in the user interface of the *Portal* component described in Section 5. In the window of the agent class behavior model editor such relations are shown as arrows between the *User command* component and respective services.

*User Interfaces*

This component represents the list and conceptual description of the user interface dialogs initiated by the agent class in question. The dialogs can either represent certain information for user or imply his response, e.g., while receiving certain data for subsequent operation. Mechanism supporting a usage of such dialogs is realized as a reusable function of the generic agent component as well.

*Pro-active model*

This component specifies possibility of agent's class to initiate an execution of any services without environmental impacts like, e.g., receiving messages or user commands. The component specifying is reduced to listing rules like *"When ... if ... then ..."*. The first condition (*When*) specifies either time instants or event initiating checking of the second condition (*if*). The second condition specifies the agent class mental state in which the service (state machine) indicated in the third part of the rule (*then*) has to be executed. The component specifying is carried out through the auxiliary dialog, and in the window of agent class behavior model editor such relations are shown as arrows between the *Pro-active model* component and the respective services. Mechanism providing for checking of these rules and initiating of the services is realized as a reusable function of the generic agent component as well.

*Environment events and Effects*

If agent is operating within an external environment then it can receive data from any external equipment (e.g., sensors) or/and their control. In this case it is necessary to specify interaction between agent class and external equipment, and the Environment events and Effects components are used then. These components specifying consists in 1) listing of interaction acts, 2) their mapping to the respective services, and 3) specifying the interface trough which data exchange is executed. Carrying out the latter task is reduced to specifying of mechanisms supporting information receiving from and sending to external equipment and algorithms of such information processing. Such mechanisms are rather application dependent and therefore they have to be specified "ad hoc", however, the generic agent component includes some reusable solutions which can be used for doing it.

Developing scenarios of the services execution implies specifying two kinds of relations: relations between components described above and services, and relations between services. The latter kind of relations is used to specify the following two cases:

- synchronous initiation of nested services, and
- asynchronous initiation of any service.

Synchronous initiation of a nested service *B* by a service *A* means the following scenario of their execution. Execution of the service A is interrupted after initiation of the service *B*, and continued after the service *B* accomplishment. At that the service *A* can use the results of the service *B* execution; the service can be considered as nested one in the following two cases: 1) when it provides for the execution of some other services, and 2) when it provides for the operation of an agent class related to any protocol. Asynchronous initiation of the service *B* by the service *A* can be considered if the results of the service *B* execution are not required for the service *A* execution. At that if some services are triggered by any service asynchronously then they will be executed in parallel.

The following relation existing between liveness properties of a role model in Gaia methodology [21] and graphic notation used for specifying of the agent class behavior mode l are worth noting. On the one hand, "the atomic components of a liveness expression are either *activities* or *protocols*" [21]. On the other hand, components that are used for specifying of an agent class behavior model denote *protocols, services* (some of them associated with activity execution), *acts of agent class interaction with user* and *with external equipment.* At that, a liveness expression includes the operators: *"x followed by y", "x or y occurs", "x and y interleaved",* which can be specified via respective kinds of relations between components and services in the agent class behavior model. All above described relations between the liveness expressions and the agent class behavior models are used at the design stage of MAS development, and allow to account for the results received at the analysis stage.

## 4.5 State Machines Editor

Specification of the agent class services is carried out in terms of state machines. This comprises the following activities: (1) specification of the set of state machine states; (2) specification of transitions and conditions determining the transition choice depending on the agent current state; (3) specification of the state machine (agent) behavior in each particular state.

The first and second activities are executed by a designer of MAS. These activities are supported by graphic editor given in Fig.9. The third activity, on the contrary, is executed by a programmer trough respective dialogs.

It should be noted that agent class behavior model, state machines and their states can be considered as different levels of description of agent class behavior. At that, agent class behavior model (Section 4.4) aims at describing agent class's behavior on the whole. State machines are used to describe partial scenarios of agent classes, and states of state machines are used to describe partial functions executed within respective scenarios. Thus, the editor of state machines can be considered the graphic editor of partial behavior scenarios of agent classes.

Specification of state machine can use specific functions that perform actions of

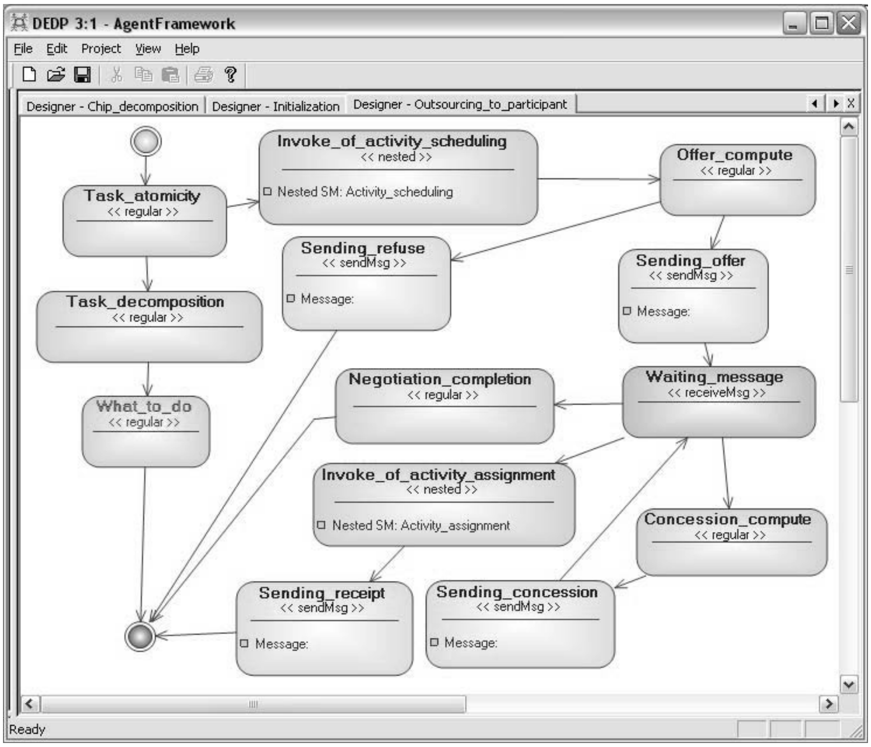


**Fig. 9.** State machine editor

several specific classes, such as processing of one or more input messages, sending one or several messages, waiting for a response, invoking asynchronous process, initiating other state machine, etc. Specification of the functions of a kind can be based on the use of respective reusable components. Since functions in MASDK are described in terms of partial states, the environment includes several pre-defined different templates of states that are used to specify functions of respective classes. States specified according to different templates are highlighted in the window of an editor by different colors (Fig.9).

Initial sketch of each state machine can be developed automatically. Component that carries out this activity uses as input data solutions developed at the previous stages of system development. In particular, it analyzes agent class behavior model, and if the state machine under development is mapped to a protocol, analyzes specification of this protocol and generates states aimed at processing the protocol's communication acts. At the following step, a designer develops in detail the above initial sketches of the state machines inserting auxiliary states and updating transitions scheme.

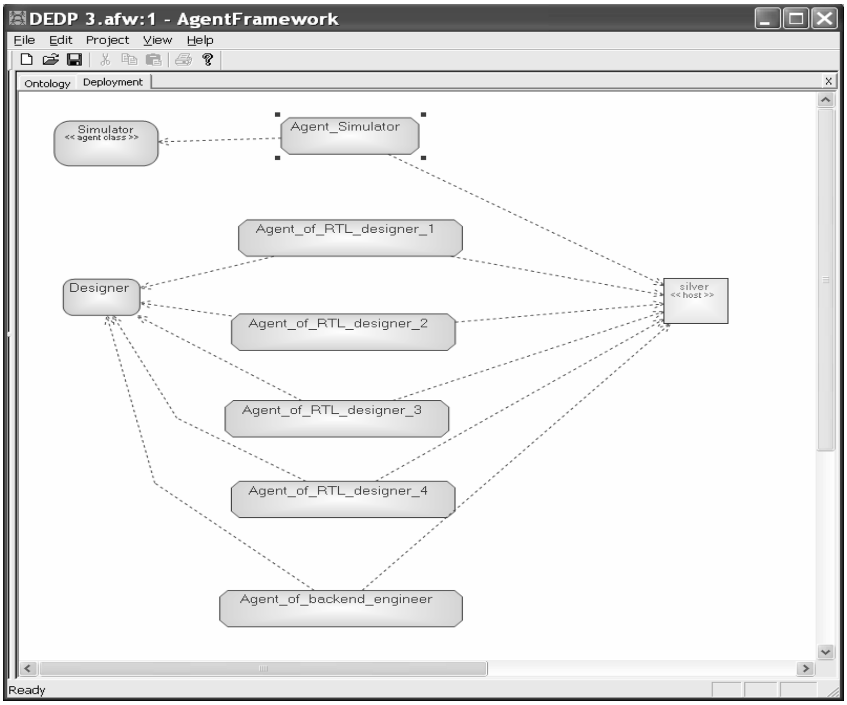


**Fig. 10.** MAS configuration editor

## 5 Deployment

Deployment of MAS consists in carrying out the following functions:

- Specification of the MAS configuration,
- Specification of initial states of agent mental models, and
- Generation and installation of software agents in the computer network according to the MAS configuration.

MAS configuration is specified using graphic editor given in Fig.10. This editor is used for specification of the following data about agents: (1) agent's unique name; (2) indication of agent class, whose instance is the software agent, and (3) the host name and its Internet address where the software agent in question has to be situated. For example, Fig.10 shows a configuration where six agents are instances of two agent classes (one agent of the first class and five agents of the second class). The configuration assumes that agents are situated in a single host.

It is important, that while having a set of agent classes developed an arbitrary number of different MAS configurations can be generated. They can differentiate in the number of agents of different classes and their allocation within Internet.

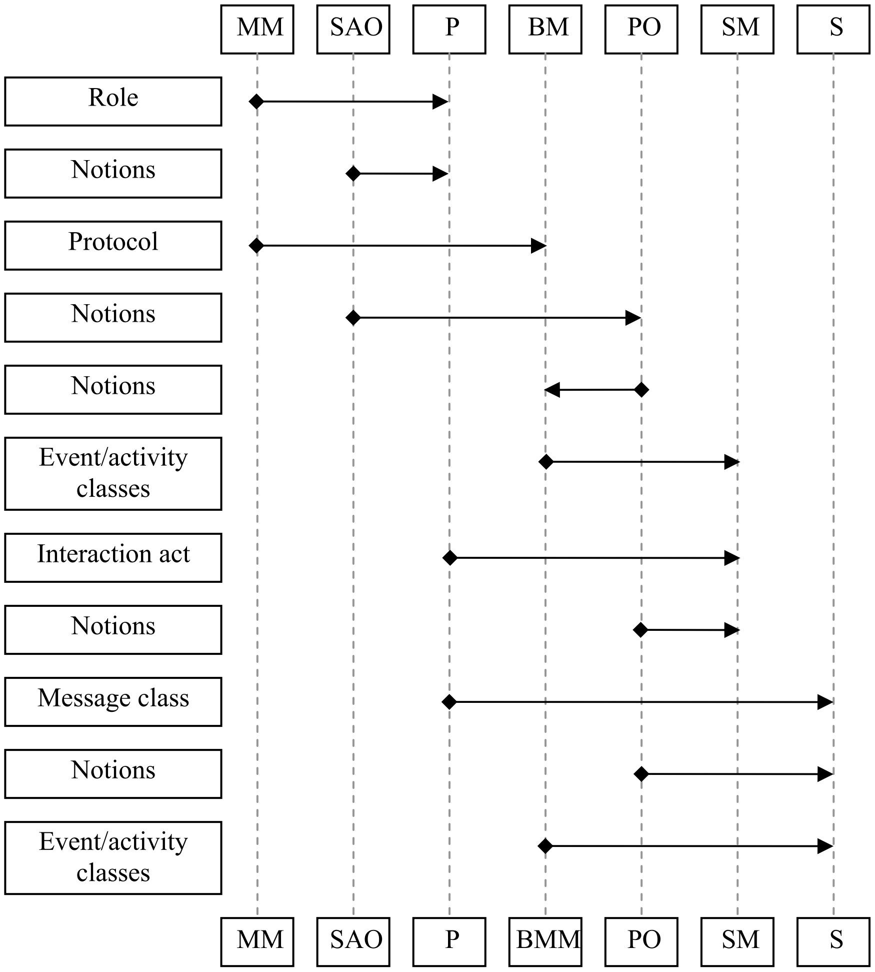
Specification of initial states of agent mental models is the next step of MAS implementation preceding its deployment. At this step, data base of each agent destined for storing of its mental model is filled in, thus, forming initial state of its mental model. It is necessary to recall that the scheme of the aforementioned data base is given in terms of notions of private ontology of the respective agent class. The above activity can be carried out either through invariant user interface, which is not depended from a agent class storage schemes, or through interfaces tuned to peculiarities of the MAS in question, i.e., tailored for them. The last version is preferable when the number of private ontology notions and/or relations between them is too large.

Executable code of the MAS of determined configuration is generated through special functional component of *Software agent builder*. Generation procedure results in filling in the initial states of software agents mental models.

Before the deployment of MAS (installation of the software agents) within the particular computer network, communication platform has to be installed. For this purpose, a reusable component of MASDK called *Portal* is installed in all the computers where MAS is deployed. Installation of *Portal* is supported by a special function of the MASDK environment.

Communication platform is destined to support the message passing (both local and remote) between MAS agents. It is important that the same communication platform can support operation of several multi-agent systems. Messages are transmitted between agents by portals these agents run on. TCP/IP is used to pass messages between agents. A message is represented as an XML with a proprietary structure. At the moment not an external message transmission protocol is supported.

*Portal* is provided with a user interface (Fig.11), that supports the following functions:



Legend:
*MM* – mental model of MAS, *SAO* –shared application ontology,
*P* – protocol, *BM* – agent class behaviour model,
*PO* – private ontology of an agent class, *SM* – state machine, *S* – state of SM

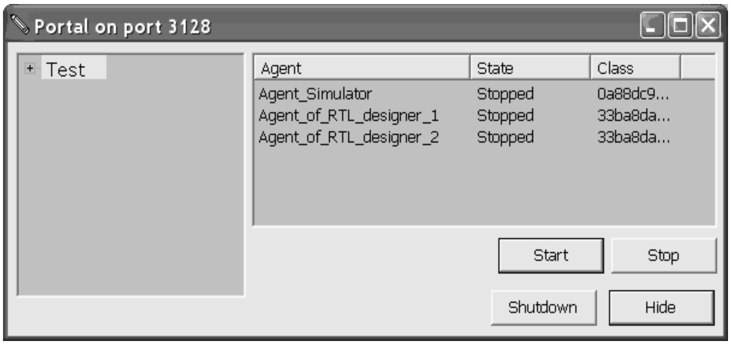
**Fig. 11.** Scheme of interrelations between components

- Initiating and shutting down the agents of a selected MAS;
- Visualization of the available user commands of operating agents and activation of their performance.

## 6 Maintenance of MAS Consistency

An important problem of MAS development is the consistency maintenance. If consistency of the MAS project components is not maintained in the development process then total development efforts and costs can considerably increase. It should be noted that the MAS design processes can potentially be well structured, thus, allowing development of special mechanisms intended for consistency maintenance. Such a possibility is based on the use and evolution of abstract notion classes utilized in the methodology. Particularly, the list of these notion classes within MASDK environment includes: role, protocol (communication acts, classes of messages), agent class, auxiliary components of agent class behavior model (see subsection 4.4), state machine (state, specialized state class), ontology notions. The use of these abstract notions basically allows regarding the design process as specification of instances of the above abstract notions and establishing different interconnections between them. While establishing such interconnections, it is necessary to meet a certain set of requirements and constraints that are independent of application domain. Scheme of main interconnections between components used in MASDK is depicted in Fig.12. This provides a possibility to well maintain the MAS consistency.

Within the MASDK 3.0 environment a special component called *Master* is used to practically exploit the above opportunity. It operates in two modes: (1) *Consistency checking* and (2) *Design mastering*. In the first mode, *Master* provides a designer with the list of consistency violations of various kinds, e.g., "*Communication act of the protocol <name_of_the_act> is not provided with the respective function on the agent class side*". While operating in the second mode, *Master* supports the necessary ordering of MAS design processes in the top-down style. At certain phases of design process this mode of operation allows *Master*, while accounting for already produced



| Agent | State | Class |
|---|---|---|
| Agent_Simulator | Stopped | 0a88dc9... |
| Agent_of_RTL_designer_1 | Stopped | 33ba8da... |
| Agent_of_RTL_designer_2 | Stopped | 33ba8da... |

**Fig. 12.** User interface of the Portal component

solutions, to automatically generate certain new solutions or their parts. For instance, if behavior model of an agent class is specified, *Master* can initiate generation of the respective set of states of state machine implementing the above behavior model.

## 7 Related Work

There are many tools for MASs development and each tool has its own features. We selected only tools that support visual designing and development. In the following sections we compare MASDK against 4 most known for us tools meeting the mentioned criterions.

### 7.1 agentTool

agentTool [11] is a tool for analyzing, designing and implementing multi-agent systems by MaSE methodology. In contrast to MASDK, the analysis phase in agentTool starts from defining goals and use cases that should help to discover roles in the target MAS. MASDK doesn't have analogues to goals and use cases. In agentTool after that each role is assigned goals it is responsible for and tasks are defined to achieve these goals. MASDK doesn't have an analogue to a task.

Then in both tools protocols are defined, but in agentTool only two roles can participate in a protocol. In contrast to agentTool, in MASDK a protocol is detailed down to communication acts. In agentTool it is also possible to define communication acts but only for agent classes irrelatively of any protocol. In both tools agent classes are defined by roles they play.

Then in agentTool each agent class is refined by defining components it consists of. Those components as communication acts are specified using a finite state automaton. In MASDK this step is performed in "SM Manager" editors that have more formal structure and advanced facilities (see details in the corresponding chapter).

At the end agentTool generates Java classes for agents and communication acts, which have to be filled in with implementation code. MASDK has built in editors for entering all required implementation code in C++ and allows to generate source code in C++ and to build executable modules. Also, MASDK has facilities to install a built MAS.

Thus from our point of view agentTool has more expressive capabilities for analyzing and designing but those for implementation are incomplete. MASDK allows to develop a MAS ready for running but relationships between designing and implementation details aren't so clear.

## 7.2 PTK

PTK supports MAS development by PASSI [8] methodology. It is implemented as a Rational Rose plug-in and uses UML for step-by-step designing and development multi-agent systems.

PTK has good facilities to support all stages of MAS development. Development in PTK starts from problem domain analysis and from functional decomposition of the target MAS. Then roles, protocols and agent classes are discovered. In contrast to MASDK, PTK has good facilities for the analysis of functionality of the target MAS and for distribution of functions between roles and agents.

An ontology in PTK is described by two diagrams: Domain Ontology and Communication Ontology. For each communication user has to define an ontology, a language and a protocol. In MASDK interactions between agents are fully described for roles these agents play.

Every agent in PTK is described as a class. Agents' behavior is displayed in one or more activity diagrams. However those diagrams have no affect on the classes. In MASDK agents' structure and behavior are specified using state machines which are automatically transformed into executable modules. PTK also delivers template agents with corresponding facilities. Finally PTK generates classes for agents, tasks and ontology which should be completed in the Pattern Repository tool. Agents created in PTK can be executed on FIPAOS or JADE.

From our point of view the design phase in PTK is more clear and useful for unskilled users but MASDK provides with more convenient facilities for implementation and deployment.

## 7.3 JACK

JACK [18] is a full-scale environment for MAS development using Java. It supports MAS development accordingly with BDI agent's model.

Analysis and design is performed in this tool using BDI concepts: Capability, Event, Plan, Agent. However there is no considerable distinction between analysis and design stages which can be done iteratively like in MASDK, because both systems support integrity. In contrast to MASDK, JACK was designed for reactive intelligent agents development. It allows to create variants of a plan to react on events. Those plans are used by the engine implemented in the generic agent for searching a way to reach a goal. In MASDK State Machine is an analogue of the plan but there is no an analogue to Capability (MASDK has some properties of Capability which are in the Role concept). MASDK is more useful when the system behavior and agents coordination with detailed description of protocols should be specified explicitly. There are no protocols in JACK but messages are treated as a kind of events. Both tools support full specification of implementation details.

Thus both systems support full development cycle with iterative modifications. MASDK is more useful for systems with a strict coordination but JACK for reactive systems. Comparing to JACK, MASDK has more advanced system of graphic editors but language constructions in JACK are more elaborated.

### 7.4 Zeus

Zeus [9] is mostly intended for development of MASs concerning planning and especially distributed planning problems. A proprietary methodology with FIPA protocols support is used in Zeus.

Development of a MAS in Zeus starts with Ontology description. Zeus allows to use single inheritance between notions and specify notion attributes with simple or user-defined types (JavaObjects). Similar functionality is also implemented in MASDK.

Then in Zeus agents are specified and the ontology notions are linked to them (MASDK has Private Ontology as an analogue). Protocols (any protocol can be one of 6 predefined types) and strategies are indicated for the agents. MASDK doesn't have an analogue to the 'strategy' concept. Zeus doesn't support further detailing of agents but in MASDK there are SM-Manager and State Machine Editors for that purposes.

At the next step project tasks are specified with connection to ontology notions and with execution restrictions. MASDK doesn't have an analogue to a task.

At last Zeus generates Java source code for the target MAS. It is possible to choose target platform: UNIX or Windows.

Thus Zeus with its planning problems orientation gives an opportunity to concentrate more on a problem domain than on interactions between agents. In contrast to Zeus, in MASDK the designing phase is supported more deeply but protocols should be created from scratch using corresponding editors. Also, MASDK has no explicit analogue of the 'task' concept.

## 8 Conclusion

MASDK environment presented in the paper possesses a number of practically important advantages allowing to noticeably decrease efforts and costs associated with the development of MASs of wide range. Among them, the most important ones are the following:

1) Development process is carried out according to well grounded methodology, in our case the *Gaia* methodology, whose abstract notion classes determined at the analysis stage are further specialized and developed in depth at the stages of design and implementation.

2) Graphical style of the development process supported by a number of user friendly editors of MASDK provides for clear and easy understandable presentation of MAS and its components along the whole development process. Together with the thorough and clear methodology graphical style provides for productive and effective cooperation between designer and programmer during the whole life cycle of MAS development.

3) Due to well structured set of abstract notion classes used in analysis and design processes, the MASDK environment provides for checking and maintenance of integrity of the development results at all stages of the process.

4) Representation of interaction protocols that is a key task in any MAS specification is particularly based on solutions involving the existing experience of object-oriented approach whose main solutions are considered now as de-facto standards in information technology.

5) One of the most important ideas of MASDK is reusability of solutions. *Generic agent* library, that integrates reusable solutions/software components, practically implements the reusability idea, thus, reducing development process to the specification of application-oriented data and knowledge.

Currently the MASDK software tool is basically implemented and is validated based on development of MASs in such problem domains as information fusion and situation assessment, computer network security, agent-based business activity simulation and monitoring.

## Acknowledgment

We wish to thank European Office of Aerospace Research and Development of the USAF (EOARD) and the Russian Academy of Sciences, Department of Information Technology and Computer Science (Project 4.3) for support granted.

We also thank Cadence Design Systems Ltd. for its realistic assessment of the MASDK capabilities and its industrial perspectives as part of a joint project. It helped to determine needs and directions of further improvement of the platform and to prove the advanced solutions implemented.

## References

1. Agent UML: http://www.auml.org/
2. Bauer, B., Muller, J. P., and Odell, J.: Agent UML: A Formalism for Specifying Multiagent Interaction. In: Ciancarini, P. and Wooldridge, M. (eds): Agent-Oriented Software Engineering, Springer-Verlag, Berlin, (2001) 91-103
3. Bellifemine, F., Caire, G., Trucco, T., and Rimassa, G.: Jade Programmer’s Guide. JADE 2.5 (2002) http://sharon.cselt.it/projects/jade/
4. Bernon, C., Gleizes, M.P., Peyruqueou, S., and Picard, G.: Adelfe, a methodology for Adaptive Multi-Agent Systems Engineering. In: Third International Workshop “Engineering Societies in the Agents World” (ESAW-2002), Madrid, (2002)
5. Bitting, E., Carter, J., and Ghorbani, A. A.: Multiagent Systems Development Kits: An Evaluation. In: Proceedings of the 1st Annual Conference on Communication Networks & Services Research, Moncton, Canada, (2003) 80-92
6. Booch, G.: Object-Oriented Analysis and Design, 2nd ed., Addison-Wesley: Reading, MA, (1994)

7. Caire, G., Leal, F., Chainho, P., Evans, R., Garijo, F., Gomez, J., Pavon, J., Kearney, P., Stark, J., and Massonet, P.: Agent-oriented analysis using MESSAGE/UML. In: Wooldridge, M., Ciancarini, P., and Weiss, G., (editors): Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001), (2001) 101-108
8. Cossentino, M., Sabatucci, L., Sorace, S., and Chella, A.: Patterns reuse in the PASSI methodology. In: Fourth International Workshop Engineering Societies in the Agents World (ESAW'03), London, UK (2003) 294-310
9. Collis, J. and Ndumu, D.: Zeus Technical Manual. Intelligent Systems Research Group, BT Labs. British Telecommunications. (1999)
10. Dam, K. H., and Winikoff, M.: Comparing Agent-Oriented Methodologies. In: Proceedings of the Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (At AAAMAS-03), Melburn (2003)
11. DeLoach S. and Wood, M.: Developing Multiagent Systems with agentTool. In: Castelfranchi, C., Lesperance Y. (Eds.): Intelligent Agents VII. Agent Theories Architectures and Languages, 7th International Workshop, LNCS. Vol.1986, Springer Verlag, (2001)
12. DeLoach, S. A., Wood, M. F., and Sparkman, C. H.: Multiagent systems engineering. In: International Journal of Software Engineering and Knowledge Engineering, 11(3), (2001) 231-258
13. FIPA-OS: A component-based toolkit enabling rapid development of FIPA compliant agents. http://fipa-os.sourceforge.net/
14. Giunchiglia, F., Mylopoulos, J., and Perini, A.: The Tropos software development methodology: Processes, Models and Diagrams. In: Third International Workshop on Agent-Oriented Software Engineering, Jula (2002)
15. Gorodetski, V., Karsaev, O., Kotenko, I., and Khabalov, A.: Software Development Kit for Multi-agent Systems Design and Implementation. In: Dunin-Keplicz, B., Navareski, E. (Eds.): From Theory to Practice in Multi-agent Systems. Lecture Notes in Artificial Intelligence, Vol. # 2296, (2002) 121-130
16. Gorodetski, V., Karsaev, O., and Konushi, V.: Multi-Agent System for Resource Allocation and Schedulling. In: Lecture Notes in Artificial Intelligence, Vol. # 2691, (2003) 226-235
17. Gorodetsky, V., Karsaev, O., and Samoilov, V.: Multi-agent Technology for Distributed Data Mining and Classification. In: Proceedings of the IEEE Conference Intelligent Agent Technology (IAT-03), Halifax, Canada, (2003) 438-441
18. Jack. Jack intelligent agents – version 3.1, agent oriented software. Ltd., Australia, http://www.agent-software.com.au .
19. Padgham, L. and Winikoff, M.: Prometheus: A pragmatic methodology for engineering intelligent agents. In: Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies, Seattle, (2002) 97-108
20. Reticular Systems Inc: AgentBuilder An Integrated Toolkit for Constructing Intelligent Software Agents. Revision 1.3. (1999) http://www.agentbuilder.com/.
21. Wooldridge, M., Jennings, N.R., and Kinny, D.: The Gaia Methodology for Agent-Oriented Analysis and Design. In: Journal of Autonomous Agents and Multi-Agent Systems, Vol.3. No. 3 (2000) 285-312
22. Woldridge, M.: Agent-based software engineering, In: IEEE Proc. Software Eng, 144(1), (1997) 26-37
23. Woldridge, M., and Jennings, N. R.: Pitfalls of agent-oriented development. In: Proc. Second Int. Conf. On Autonomous Agents (Agents 98), Minneapolis/St Paul, MN, (1998) 385-391
24. http://www.agentbuilder.com/AgentTools/index.html.
25. http://www.agentlink.org/resources/agent-software.php

## Information about software

### Software is available in the Internet as:

( ) prototype version
( ) full fledged software (freeware), version no.:
( ) full fledged software (for money), version no.:
( ) Demo/trial version
(x) not (yet) available

### Contact person for question about the software:

Name: Oleg Karsaev
email: ok@mail.iias.spb.su

Oleg Karsaev
Laboratory of Intelligent Systems
St. Petersburg Institute for Informatics and Automation
14 Line, 39, St. Petersburg, 199178
Russia
email: ok@mail.iias.spb.su