Chapter 5

Computational Agents in Complex Decision Support Systems

Antonio Fernández-Caballero[1] and Marina V. Sokolova[1,2]

[1] Universidad de Castilla-La Mancha, Departamento de Sistemas Informáticos
Campus Universitario s/n, 02071-Albacete, Spain
caballer@dsi.uclm.es, marina.v.sokolova@gmail.com
[2] Kursk State Technical University, Kursk, ul.50 Let Oktyabrya, 305040, Russia

Abstract. The article introduces a general approach to decision making in complex systems and architecture for agent-based decision support systems (DSS). The approach contributes to decentralization and local decision making within a standard work flow. The architecture embodies the logics of the decision developing work flow and is virtually organized as a layered structure, where each level is oriented to solve one of the three following goals: data retrieval, fusion and pre-processing; data mining and evaluation; and, decision making, alerting, solutions and predictions generation. In order to test our approach, we have designed and implemented an agent-based DSS, which deals with an environmental issue. The system calculates the impacts imposed by the pollutants on the morbidity, creates models and makes forecasts by permitting to try possible ways of situation change. We discuss some used data mining techniques, namely, methods and tools for classification, function approximation, association search, difference analysis, and others. Besides, to generate sets of administrative solutions, we develop decision creation and selection work flows, which are formed and then selected in accordance with the maximum of possible positive effect and evaluated by external and internal criteria. To conclude, we show that our system provides all the necessary steps for standard decision making procedure by using computational agents. We use so much traditional data mining techniques, as well as other hybrid methods, with respect to data nature. The combination of different tools enables gaining in quality and precision of the reached models, and, hence, in the recommendations that are based on these models. The received dependencies of interconnections and associations between the factors and dependent variables help correcting recommendations and avoiding errors.

1 Introduction

The use of agent-based intelligent decision support systems (IDSS) is important for the environmental related issues, because they allow specialists to quickly gather information and process it in various ways in order to understand the real nature of the processes, their influence on human health, and the possible outcomes in order to make preventive actions and take correct decisions. The areas these systems could help in are diverse, from the storing and retrieval of necessary records, storing and retrieval of key factors, examination of real-time data gathered from monitors, analysis of tendencies of environmental processes, retrospective time series, making short and long-term forecasting, and in many other cases [1-3].

Nowadays, in the area of agent-based systems there are a lot of applications of decision support systems (DSS) for social and ecological issues in general, and for the environmental impact upon human health assessment problem in particular [4]. To understand the current trends and to assess the ability of current agent-based intelligent decision support research it seems to be reasonable to survey the current state of the art and conclude how it is possible to optimize it.

2 Decision Support Systems for Complex Systems Study

The majority of real-life problems related to sustainable development and environment can be classified as complex composite ones, and, as a result, they have some particular characteristics, due to those, they require interdisciplinary approaches for their study. A system is an integration of interconnected (through informational, physical, mechanical, energy exchange, etc.) parts and components, which results in emerging of new properties, and which interacts with the environment as a whole entity. If any part is being extracted from the system, it loses its particular characteristics, and converts into an array of components or assemblies. An effective approach to complex system study has to follow the principles of the system analysis, which are:

- Description of the system. Identification of its main properties and parameters.
- Study of interconnections amongst parts of the system, which include informational, physical, dynamical, temporal interactions, as well, as functionality of the parts within the system.
- Study of system interactions with the environment, in other words, with other systems, nature, etc.
- System decomposition and partitioning. Decomposition supposes extraction of series of system parts, and partitioning suggests extraction of parallel system parts. These methods can be based on cluster analysis (iterative process of integration of system elements into groups) or content analysis (system division into parts, based on physical partitioning or function analysis).
- Study of each subsystem or system part, utilizing optimal corresponding tools (multidisciplinary approaches, problem solving methods, expert advice, knowledge discovery tools, etc.)
- Integration of results received on the previous stage, and obtaining a pooled fused knowledge about the system. Synthesis of knowledge and composition of a whole model of the system can include formal methods for design,

multi-criteria methods of optimization, decision-based and hierarchical design, artificial intelligence approaches, case-based reasoning and others, for example, hybrid methods.

It is obvious, that a DSS structure has to satisfy the requirements, imposed by specialists, and characteristics and restrictions of the application domain. On Fig. 1 there is a general workflow of a decision making process, which is embodied in a DSS. The traditional "decision making" workflow includes the preparatory period, development of decision and, finally, decision making itself and its realization.

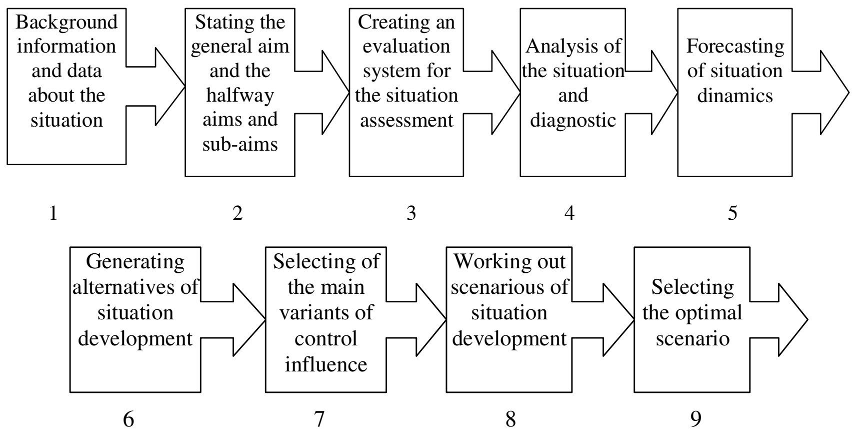
Background information and data about the situation
Stating the general aim and the halfway aims and sub-aims
Creating an evaluation system for the situation assessment
Analysis of the situation and diagnostic
Forecasting of situation dinamics
1
2
3
4
5
Generating alternatives of situation development
Selecting of the main variants of control influence
Working out scenarious of situation development
Selecting the optimal scenario
6
7
8
9

Fig. 1. The general workflow of a decision making process

In accordance with Fig. 2, a decision can be seen as the result of local decisions, alternatives, which satisfy selection criteria. The complexity increases in case if all these spaces have a composed organization. In the simplest case, possible alternatives are independent, but they can be grouped into clusters, or form hierarchies; decisions can consist of the best optimal alternative, but can also be formed as a result of combination (linear, non-linear, parallel, and so on) of alternatives, and their subsets and stratifications; criteria can be both independent or dependent, and, commonly, hierarchically organized.

Our approach towards DSS for complex system is based on the general DSS structure discussed in section 3. The main components of the DSS, which are (a) the user interface, (b) the database, (c) the modeling and analytical tools, and (d) the DSS architecture and network, have been determined for special features and characteristics of possible application domains. The most important difference is that the DSS is realized in form of a multi-agent system, and the agents provide system functionality and realize organizational and administrative functions.

DSS organization in form of a MAS facilitates distributed and concurred decision making, because the idea of the MAS serves perfectly to deal with difficulties of a complex system. A MAS, which can be described as a community of

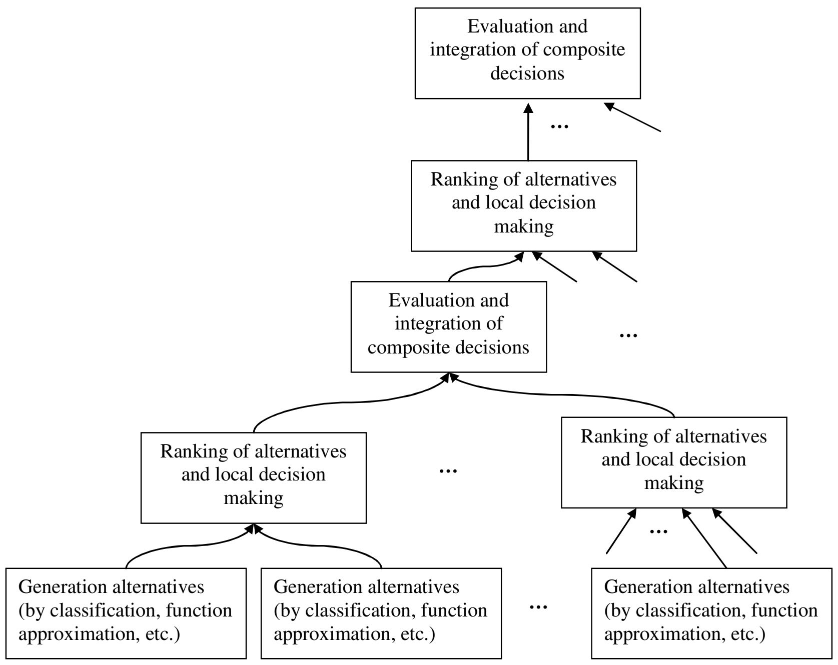
Evaluation and integration of composite decisions
Ranking of alternatives and local decision making
Evaluation and integration of composite decisions
Ranking of alternatives and local decision making
Ranking of alternatives and local decision making
Generation alternatives (by classification, function approximation, etc.)
Generation alternatives (by classification, function approximation, etc.)
Generation alternatives (by classification, function approximation, etc.)
...

Fig. 2. Composite decision as a result of compositions between possible decisions, alternatives, with respect to selection criteria

intelligent entities – computational agents, offers solutions because of the agency properties, which are: reactivity (an agent responds in a timely fashion to changes in the environment); autonomy (an agent exercises control over its own actions); goal-orientation (an agent does not simply act in response to the environment, but intents to achieve its goals); learning (an agent changes its behavior due to its previous experience), reasoning (the ability to analyze and make decisions), communication (an agent communicates with other agents, including external entities), and mobility (an agent is able to transport itself from one machine to another) [5-6].

3 Decision Support Systems and Their Characteristics

There are many definitions of what a DSS is; for example, one is that decision support systems are a specific class of computerized information systems that support business and organizational decision-making activities. A properly-designed DSS is an interactive software-based system intended to help decision makers in compiling useful information from raw data, documents, personal knowledge, and/or business models to identify and solve problems and make decisions. Bonczek et al. [7] define a DSS as a computer-based system consisting of three interacting components: a language system, a knowledge system, and a problem-processing system. This definition covers both old and new DSS designs, as the problem processing system could be a model-based, or an expert system, or an agent-based system, or some other system providing problem manipulation capabilities. Keen [8] applies the term DSS to situations where a 'final' system can be developed only through an adaptive process of learning and evolution. Thus, he defines a DSS as the product of a developmental process involving the builder, the user and the DSS itself to evolve into a combined system.

Sprague and Carlson [9] identify three fundamental components of a DSS: (a) the database management system (DBMS), (b) the model-based management system (MBMS), and, (c) the dialog generation and management system (DGMS). Levin [10] analyses a number of works and names the following components as essential for a modern DSS: (1) models, which include multi-criteria techniques, problem-solving schemes, data processing and knowledge management; (2) analytical and numerical methods of data pre-processing and identification of problems for the preliminary stages of decision making; (3) human-computer interaction and its organization through graphic interface and others; (4) information support, communication with databases, web-services, etc. According to Power [11], academics and practitioners have discussed building DSS in terms of four major components: (a) the user interface, (b) the database, (c) the modeling and analytical tools, and (d) the DSS architecture and network. The definition of a DSS, based on Levin and Power, in that a DSS is a system to support and improve decision making, to our point of view, represents an optimal background for practical DSS creation.

3.1 Agents and Decision Support Systems

Agents and multi-agent systems (MAS) are actively used for problem solving and have recommended themselves as a reliable and powerful technique [12-14]. The "agent" term has many definitions, and, commonly, is determined as "an entity that can observe and act upon an environment and directs its activity towards achieving goals". In practice, agents are often included into multi agents systems, which can be determined as a decentralized community of intelligent entities task solvers (computational agents), oriented to some problem. The agents in a multi-agent system have several important characteristics [6]:

- **Autonomy:** the agents are at least partially autonomous.
- **Local views:** no agent has a full global view of the system, or the system is too complex for an agent to make practical use of such knowledge
- **Decentralization:** there is no central guidance in the system, and the agents use their reasoning abilities to act in accordance with internal believes.

MAS can manifest self-organization and complex behaviors even when the individual strategies of all their agents are simple. The agents can share knowledge using any agreed language within the constraints of the system's communication protocol. Example of agent communication languages are Knowledge Query Manipulation Language (KQML) or FIPA's Agent Communication Language (ACL).

There is a need of mechanisms for advertising, finding, fusing, using, presenting, managing, and updating agent services and information. Most MAS use facilitator agents to help find agents, agents to which other agents surrender their autonomy in exchange for the facilitator's services. Facilitators can coordinate agents' activities and can satisfy requests on behalf of their subordinated agents. MAS can be classified in accordance with several classifiers. There are closed and open MAS. The first ones contain well-behaved agents designed to cooperate together easily towards a global goal. The MAS, related to the second type, can contain agents that are not designed to cooperate and coordinate, but to assist the agents in working together. The most common kind of these mechanisms is for negotiations and auctions.

Weiss gives other classifications: MAS classified by the level of autonomy, of organizational type, and architecture [6]. Depending on the level of autonomy and self-orientation of every agent, MAS can vary from distributed and "independent" to supervised systems of "organizational" type, in which every agent knows the order and turn of its execution. The MAS is a kind of an informational system, and its planning and creation, actually, include the same set of tasks and works as in the general case of any software tool.

3.2 Multi-agent Planning and Design

3.2.1 Multi-agent Developing Life Cycle

Creation, deployment and post-implementation of a MAS as a software product is a complex process, which passes through a sequence of steps forming its life cycle. Every step of the life cycle process has to be supported and provided by program tools and methodologies. In case of MAS development, there is no unified approach to cover all the steps. However, there are some works dedicated to this issue [15-16]. For instance, de Wolf and Holvoet [15] have offered a methodology in the context of standard life cycle model, with accent to decentralization and macroscopic view of the process. As tools and frameworks the authors mention Jade [17], Repast [18] and an environment for the coordination of situated multi-agent systems [19].

The authors offer their methodology on the assumption that the research task has already been defined. Though, the software life cycle includes the problem definition and domain analysis stages, which cannot be omitted. The software development in case of MAS is based on the following steps [20]:

1. ***Domain Analysis*** - is related to the analysis of the project idea, problem definition, extraction of aims, creation of the goal trees, sequence of tasks and subtasks to be solved. This stage also implies domain ontology creation, which covers the problem area, the set of relations between the concepts and the rules to receive new knowledge. The work of domain areas experts is required at this stage.
2. ***Software Elements Analysis*** - this stage also deals with private ontologies creation, but now, ontologies are created for the system and its elements. The

sets of goals and tasks are related to the sets of system functions (roles), required resources (commonly in form of informational files), interactions, etc.

3. ***Specification*** - is the written description of the previous stages, which results in system meta-ontology creation.
4. ***Software Architecture*** - implies the abstract representation of the system to meet the requirements. The software architecture includes interfaces for computer-user communication.
5. ***Implementation (coding)*** - the iterative process of program creation.
6. ***Testing*** - program testing under normal and critical conditions.
7. ***Deployment and Maintenance*** - program application and support until the software is put into use. Sometimes some training classes on the software product are made.
8. ***End of Maintenance*** - is a final stage of the software life-cycle.

3.2.2 Ontology Representation

There are a great number of languages for ontology creation. To name some, but not all, we have: OKBC, Ontolingua/KIF, OIL, SHOE, XOL, DAML+OIL, CycL, OWL, and RDF. The creation of XML (eXtensible Markup Language) appeared to be a visible advantage towards knowledge representation in the Web. The XML gives the users a range of possibilities to create their own logical systems of data representation, determining tags, structural elements and their relations. All the connections between tags can be settled and stored in DTD (Document Type Definition) or XML schema document. The modes of data representation in XML documents are defined in XSL (eXtended Style Language) files. Though XML permits to organize and to structure data representation, it lacks possibilities to represent their semantics, because in XML there is no standard for tags and their relations definition.

3.2.3 Meta-ontology planning

According to Guarino et al. [21], an ontology can be understood as an intentional semantic structure which encodes the implicit rules constraining the structure of a piece of reality. There are a number of approaches to ontology creation, mostly induced by the specificity of the domain of interest and the nature of the tasks to solve (e.g. [22]), from which we can induce and convert to our aims an algorithm of distributed ontology creation:

1. Situation description in natural language.
2. Vocabulary creation (extraction of concepts describing the situation).
3. Taxonomy creation.
4. Distributed meta-ontology structure creation.
5. Domain of interest ontology statement.
6. Description of tasks to solve and creation of the respective private ontology.
7. Description of MAS roles, agents and creation of the system architecture ontology.
8. Description of agent ontology.

9. Agent environment ontology statement by specifying interaction and communication protocols.
10. Ontologies mapping.
11. Data Bases filling for a MAS.
12. Data Sources delivering to agents.

When briefly studying the steps of a given algorithm, it is worth noting that step 1 - problem description - serves for better understanding the aims of the research and structure of the functionality of the situation. This initial analysis helps defining concretely the problem at hand and recovering the concepts, their characteristics and relations to examine. On this stage, expert information, which is supplemented by statistical data and multimedia references related to the problem, is used.

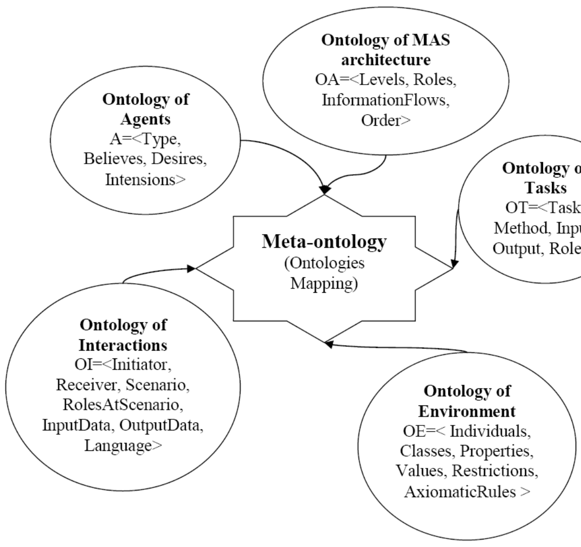
Ontology of MAS architecture
OA=<Levels, Roles, InformationFlows, Order>
Ontology of Agents
A=<Type, Believes, Desires, Intensions>
Ontology of Tasks
OT=<Task, Method, Input, Output, Roles>
Meta-ontology
(Ontologies Mapping)
Ontology of Interactions
OI=<Initiator, Receiver, Scenario, RolesAtScenario, InputData, OutputData, Language>
Ontology of Environment
OE=< Individuals, Classes, Properties, Values, Restrictions, AxiomaticRules >

Fig. 3. Components of the meta-ontology

The consequentially following task (2) is the creation of a vocabulary, which includes the necessary and sufficient information about the concepts. The further step 3 consists in adding a set of relations (including hierarchical ones) between the concepts to a vocabulary, which results into a taxonomy. As in our work we use the inductive method of ontology creation, then, on step 4 we determine the general structure of the meta-ontology (see Fig. 3) and extract the main functionally and semantically separated components. On steps 5 to 8 we create private ontologies for the extracted components of the meta-ontology, namely domain of interest, MAS architecture, tasks, agents and interactions. At steps 9 and 10 the private ontologies are mapped together. Finally, we fill data bases for a MAS (11) and deliver the real data to agents (12). In the following part of the article the distributed meta-ontology and the private ontologies, as well as the mapping procedure, are described in detail.

To provide the ontological basis as for the domain of interest, as well as for the MAS structure and organization, the meta-ontology creation framework, which maps together private ontologies, was developed. The Distributed Meta-Ontology is obtained as a result of private ontologies mapping, and is pooled by their common use and execution. This is achieved at step 10 of the algorithm proposed before. The shared ontological dimension, filled with the data, provides agents with correct addressing to proper concepts and synchronizes the MAS functionality.

3.2.4 Frameworks of Multi-agent Systems Planning

Domain Analysis and Software Elements analysis steps, noted in the MAS developing algorithm, can be made through domain analysis and ontology creation, using software products for knowledge representation, described in the previous part. Thus, "Software elements analysis" needs information about MAS functionality and taxonomy. Here we can use one of the several frameworks, existing and used in scientific practice. Some of the most frequently used, but not limited, are: MaSE [23], Gaia [24], Agent ULM [25], Prometheus [26], Tropos [27], INGENIAS [28], and some others.

Agent-Oriented Software Development is one of the recent contributions to the field of Software Engineering. To date numerous methodologies for agent-oriented software development have been proposed in the literature. However, their application to real-world problems is still limited due to their lack of maturity. Evaluating their strengths and weaknesses is an important step towards developing better methodologies in the future. MAS bring some difficulties to a researcher, which are caused by task identifications, specifying sets of protocols, interactions, methods and agents behaviors. That makes software design tools more sophisticated, which operate with new concepts as agents, goals, tasks, interactions, plans, believes, etc. Methodologies offer different tools to cope with the complicity and facilitate MAS planning and design [29]. The brief review of some methodologies includes the following ones.

The Prometheus methodology defines a detailed process for specifying, designing and implementing agent-oriented software systems. It consists of three phases [30]: the *System Specification* phase, which focuses on identifying the goals and basic functionalities of the system, along with inputs (percepts) and outputs (actions) [31]; the *Architectural Design* phase, which uses the outputs from the previous phase to determine which agent types the system will contain and how they will interact; and, the *Detailed Design* phase, which looks at the internals of each agent and how it will accomplish its tasks within the overall system.

System Specification. The Prometheus methodology focuses particularly on specification of *goals* [32], and on *scenario* description [33]. In addition, it requires specification of *functionalities* – small chunks of behavior – related to the identified goals. There is also a focus on how the agent system interfaces with the environment in which it is situated, in terms of percepts that arrive from the environment, and actions that impact on the environment. As part of the *interface* specification, Prometheus also addresses interaction with any external data stores or information repositories. The aspects developed in the *System Specification* phase are: specification of system goals with associated descriptors, development of a set of scenarios that have adequate coverage of the goals, definition of a set of functionalities that are linked to one or more goals and which provide a limited piece of system behavior, and, description of the interface between the agent system and the environment in which it is situated.

Architectural Design. The three aspects that are developed during the *Architectural Design* phase are: deciding on the *agent types* used in the application, describing the interactions between agents using *interaction diagrams* and *interaction protocols*, and, describing the system structure through the *system overview diagram*.

Detailed Design. In the *Detailed Design*, for each individual agent, it is decided what *capabilities* are needed for the agent to fulfill its responsibilities as outlined in the functionalities it contains. The *process specifications* to indicate more of the internal processing of the individual agents are developed. And when getting into greater detail, the capability descriptions to specify the individual *plans, beliefs* and *events* needed within the capabilities are developed. Then the views that show processing of particular tasks within individual agents are developed. It is during this final phase of detailed design that the methodology becomes specific to agents that use event-triggered plans in order to achieve their tasks.

The Gaia methodology [24] provides a full support for multi-agent system creation starting from the requirements determination, up to the detailed design. There are two phases of modeling with Gaia: analysis and design. The aim of the first stage is to understand the system structure and its description. The 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 can be easily implemented*".

The analysis phase. The analysis phase supposes the following steps:

1. Identification of the roles.
2. Detailed description of the roles.
3. Modeling interactions between the roles.

At first phase, as the requirements to the system are stated, there are two models to be created: the Roles model and the Interactions model. To create a Roles model, the developer has to understand the main purposes of the system created, analyze the organizational and functional profile of the system, which is decomposed and represented by set of played roles. The concept of "Role" is one of the key ones in Gaia methodology, as it determines a function related to some system task (or tasks), which is semantically and functionally interacts with the other roles. Role can be related to a system entity, for example, in case of human organization, a role can represent a "manager" and "seller".

The role is defined by the following attributes: responsibilities, permissions, activities and protocols. Responsibilities determine functions of the role and have aliveness properties and safety properties. Aliveness properties describe the actions and conditions that the agent will bring, by the other words, it determines the consequences of executed procedures, which will be potentially undertaken within a role. Safety properties state crucial environmental conditions, which cannot be exceeded or neglected. Permissions determine the resources and their limits for the role, and are commonly represented by information resources. For example, it can be abilities to read, change or generate information. Activities are private actions of an agent, which are executed by the agent itself, without communication with the other agents. Every role can be associated with one or more protocols, which state communications with other roles. The described attributes for every role are pooled in so-called role schemata, thus, comprising the Roles model.

The interaction model is focused on protocols description and their comprising. Protocols determine links between roles and provide the interaction within the multi-agent structure. The protocol definition includes: purpose - the brief textual description or detailed name of the protocol, which discovers the nature of interaction; initiator – the name of the role, which initiated the interaction; responder – the role with which the initiator communicates; inputs - the information, supplied by the responder during the interaction; processing – the brief description of processes realized within the protocol. Finally, in the analysis phase, there are the Roles models created, with the associated protocols, comprising the Interaction model.

The design phase. During the design phase, service, agent and acquaintance models are created. These models provide detailed description of the multi-agent system that then could be easily implemented. The Agent model relates roles to every agent type, taking into account that an agent may play one or more roles. The agents form a hierarchy in which leaf nodes correspond to roles and other nodes to other agent types. The number of agent instances is also documented; for example, the agent may be called for execution once, or *n* times, or repeated from *m* to *n* times, etc. The Services model identifies the necessary resources for every function performed by an agent. Every function (or service) has properties, which include inputs, outputs - those are derived from protocols, and pre-conditions and post-conditions, which state constraints and are derived from safety properties of related roles. The Acquaintance model is created from the Interactions and the Agent models, and serves to state communication links between the agents, and is represented by directed graphs, where each vertex of those relates to an agent, and every edge to a communication link.

3.2.5 Software Tools for Mutli-agent Systems Design and Implementation

Because of the complex nature of problems to solve, multi-agent systems become more complicated to plan and to design. There appeared new concepts such as goals, roles, plans, interactions, environment, necessary to identify system functionality, interactions between agents, mental states and behavior of the last. On the other hand, MAS have to be secure, mobile and able to cope with distributed problem solving. These put on requirements on methodologies to help designers to deal with these problems, and manage this complexity. A methodology should facilitate and support agent-based system engineering by providing solid terminology support, precise notations and reliable interactions, and general system functionality organization.

Nowadays, there are a number of methodologies for MAS planning and design, which are divided into steps, during which the system is firstly described in general terms, and then in more details, which determine the internal functionality of system entities. Two well-known methodologies were presented and discussed in the previous section. And, in this part software tools for the system coding implementation will be discussed. These can be viewed as a logical continuation of the methodologies (JACK Software tool is related with Prometheus Design Tool and MASDK is based on Gaia methodology).

JACK Development Environment (JDE) is a software package for agent-based applications development in Java-based environment JACK [35]. The JDE has a visual interface, which supports application creation. This may be done directly in JDE environment, or be imported, for example, from Prometheus Development Tool, a graphical editor which provides agent systems design in accordance with its associated methodology Prometheus [35]. The JDE enables building applications by providing a visual representation of the system components in two modes: agent mode and team mode.

Jack, written in Java, provides object-oriented programming for the system, encapsulating the desired behavior in modular units so that agents can operate independently. JACK intelligent agents are based on the Believe-Desire-Intention model, where autonomous software components (the agents) pursue their given goals (desires), adopting the appropriate plans (intentions) according to their current set of data (beliefs) about the state of the world.

Hence, a JACK agent is a software component that has (a) a set of beliefs about the world (its data set), (b) a set of events that it will respond to, (c) a set of goals that it may desire to achieve (either at the request of an external agent, as a consequence of an event, or when one or more of its beliefs change), and, (d) a set of plans that describe how it can handle the goals or events that may arise. JACK permits the creation of multiple autonomous agents, which can execute in agent and in team mode within a multi-agent system. MAS creation can be realized using a graphical interface. JACK extension to Team mode permitted Teams Models to be treated as peers, and introduces new concepts as team, role, team data and team plan, which required to wide semantics of some elements, and to appear team reasoning entity, *knowledge and* internal coordination of the agents within the team. The key concept, which appears here, is the *role* concept. A role defines the means of interacting between a containing team (*role tenderer*) and a contained team (*role performer* or *role filler*). In JACK Team mode each team has its lifetime, which is divided into two phases: first phase is for setting up an initial *role obligation structure* and the second phase constitutes the actual operation of the team. In addition to the agent believes, in team mode, knowledge can be "propagated" over the team members.

Prometheus Development Kit permits creation of the skeleton code for its later implementation in JACK, which facilitates the stages of MAS planning and coding. Actually, JACK teams can be used for complex distributed system modeling and problem solving.

Java Agent DEvelopment Framework (JADE) is a software Framework fully implemented in Java language [17]. Developers position JADE as "a middleware for the development and run-time execution of peer-to-peer applications which are based on the agent paradigm and which can seamless work and interoperate both in wired and wireless environment." JADE facilitates the development of distributed applications composed of autonomous entities that need to communicate and collaborate in order to achieve the working of the entire system.

On the one hand, JADE is a run-time system for FIPA-compliant MAS, which supports application agents agree with FIPA-specification. On the other hand, JADE provides object-oriented programming through messaging, agent life-cycle managing, etc. Functionally, JADE provides the basic services necessary to distributed peer-to-peer applications in the fixed and mobile environment. Each agent can dynamically discover other agents and is able to communicate with them directly. Each agent is identified by a unique name and provides a set of services, manage them, can control its life cycle and communicate with others.

JADE is a distributed platform, which comprises one or more agent containers, supported by Java Virtual Machine (JVM) each, and JVM provides a complete run time environment for agent execution and allows several agents to concurrently execute on the same host. The configuration can be controlled via a remote graphic-user interface. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required. JADE has two types of messaging: inter-platform and intra-platform (interacting agents are inside the same platform). Messaging, realized in Agent-Communication Language (ACL), is presented in form of queue, which can be accessed via a combination of several modes: blocking, polling, timeout and pattern matching based. JADE is completely implemented in Java language and the minimal system requirement is the version 1.4 of JAVA (the run time environment or the JDE).

Multi-agent System Development Kit (MASDK) is a relatively new methodology, created in the Laboratory of Intelligent Systems, St. Petersburg Institute for Informatics and Automation of the Russian Academy of Sciences [4,36]. The software tool provides support for the whole life cycle of MAS development. As a terminological foundation, the authors use the Gaia methodology.

MASDK 3.0 software tool consists of the following components: (1) system kernel which is a data structure for XML–based representation of applied MAS formal specification; (2) integrated set of the user friendly editors supporting user's activity aiming at formal specification of an applied MAS under development at the analysis, design and implementation stages; (3) library of C++ classes of reusable agent components constituting what is usually called Generic agent; (4) communication platform to be installed in particular computers of a network; and (5) builder of software agent instances responsible for generation of *C++* source code and executable code of software agents as well as deployment of software agents over already installed communication platform.

MASDK includes three editors, which act on each of the three levels. The editors of the first one correspond to the Gaia´s analysis phase and are dedicated to ontology determination, roles extraction and determination of protocols and interactions between the agents. The editors of the second level support the design activities and primarily aim at specification of agent classes. They include agents which determine behavior, agent ontologies, functions and plans. The editors of the third level support implementation stage of applied MAS and particular components and lists of agents instances of all classes with the references to their locations (hosts names), and initial states of agent believes. The next stage is correspondent to the design phase of the Gaia methodology, where the developer fills generalized MAS structural entities with internal components, which are the following ones: (1) invariant (reusable) component called Generic Agent, (2) meta-model of agent class's behavior, (3) a multitude of functions of agent class represented in terms of state machines, and, (4) library of specific auxiliary functions The applied MAS specification produced by designers exploiting the above editors is stored as an XML file in the system kernel. This specification, including a set of particular components and functions implemented in C++, and Generic Agent reusable component form the input of the software agent builder generating automatically software code based on XSLT technology.

4 General Approach for Multi-agent System Creation

4.1 Information Change

Large amounts of raw data information describe the "environment - human health" system, but not all the information can be of use though. For the situation modeling we orient to factual and context information, presented in data sets and we use computational agents to extract it. So, the information transforms from the initial "raw" state to the "information" state, which suggests organized data sets, models and dependencies, and, finally, to the "new information" which has a form of recommendations, risk assessment values and forecasts. The way in which the information changes, is given in Fig. 4.

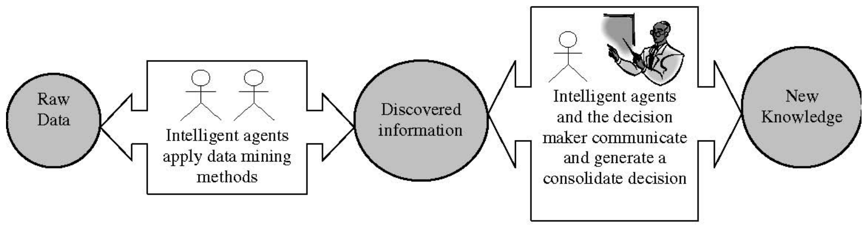
Raw Data
Intelligent agents apply data mining methods
Discovered information
Intelligent agents and the decision maker communicate and generate a consolidate decision
New Knowledge

Fig. 4. The information transformation, which changes from weakly organized and heterogeneous view into the form of knowledge

The hidden information is discovered by agents, but for new information construction not only intelligent agents, but knowledge of decision maker or expert are involved. The agent-based decision support system (MAS) we are creating provides these information changes. The process of information change, shown on

Fig. 4, corresponds to the MAS life cycle flow, which, in case of MAS, counts the following steps:

1. **Domain Analysis** - is related to the analysis of the project idea, problem definition, extraction of aims, creation of goal trees, sequencing of tasks and subtasks to be solved. This stage also implies the domain ontology creation, which covers the problem area, the set of relations between the concepts and the rules to incorporate new knowledge. The experience of domain area experts is required on this stage.
2. **Software Elements Analysis** - this stage also deals with private ontologies creation; but now ontologies are created for the system and its elements. The sets of goals and tasks are related to the sets of system functions (roles), required resources (commonly in form of informational files), interactions, and so on.
3. **Specification** - is the written description of the previous stages, which results in system meta-ontology creation.
4. **Software Architecture** - implies the abstract representation of the system to meet the requirements. The software architecture includes interfaces for human-computer communication.
5. **Implementation** - the iterative process of program creation.
6. **Testing** - program testing under normal and/or critical conditions.
7. **Deployment and Maintenance** - program application and support until the software is put into use. Sometimes some training classes on the software product are made.
8. **End of Maintenance** - is the final stage of the software life cycle.

The workflow of tasks, which has to be solved for information integration (see Fig. 2) contains four sequential states of data transformation: (1) initial heterogeneous data sources, (2) storages of extracted data, (3) mapped (fused) meta-data, (4) shared global ontology of the problem area (domain ontology) and three flows/processes, which provide and organize the transformations: (i) data retrieval and extraction, (ii) data mapping (fusion), (iii) filling in the ontology of the problem area (domain ontology).

4.2 Multi-agent System Organization and Architecture

We have implemented an agent-oriented software system dedicated to environmental impact assessment. The system receives retrospective statistical information in form of direct indicator values - water pollution, solar radiation - and in form of indirect indicator values - types and number of vehicles used, energy used annually and energy conserved, types and quantity of used fuel, etc. The indirect indicators are utilized in accordance with ISO 14031 “Environmental Performance Evaluation” standard in order to estimate air and soil pollution [37]. The population exposure is registered as number of morbidity cases with respect to International Statistical Classification of Diseases and Related Health Problems, 10th review (ICD-10) [38].

In order to provide the system design we decided to use the Prometheus Development Tool (PDT), which provides a wide range of possibilities for MAS planning and implementation: the system architecture, the system entities, their internals and communications within the system and with outer entities. The most important advantages of PDT are an easy understandable visual interface and the possibility to generate code for JACK™ Intelligent Agents, which is used for MAS implementation, verification and maintenance.

The initial analysis of the system has resulted in obtaining and describing the system roles and protocols. There, the proposed system is logically and functionally divided into three layers; the first is dedicated to meta-data creation (information fusion), the second is aimed to knowledge discovery (data mining), and the third layer provides real-time generation of alternative scenarios for decision making.

The goals drawn in Fig. 5 repeat the main points of a traditional decision making process, which includes the following steps: (1) problem definition, (2) information gathering, (3) alternative actions identification, (4) alternatives evaluation, (5) best alternative selection, and, (6) alternative implementation. The first and the second stages are performed during the initial step, when the expert information and initial retrospective data is gathered, the stages 3, 4 and 5 are solved by means of the MAS, and the 6th stage is supposed to be realized by the decision maker. Being implemented by means of the Prometheus Design Tool, the Analysis Overview Diagram of the MAS enables seeing the high-level view composed of external actors, key scenarios and actions (see Fig. 5). The proposed MAS presupposes communication with two actors. One actor is named as “Expert” and it embodies the external entity which possesses the information about the problem area -in more detail, it includes the knowledge of the domain of interest represented as an ontology -and delivers it through protocol ReturnEI to the MAS.

The data source, named “The CS Results” stores the results of the simulation and forms a knowledge base (KB). Through the *Simulate Models* scenario user interacts with the KB, and gets recommendations if they have been previously simulated and stored before, or creates and simulated the new ones. As a result of the interaction within the *FuseHeterogeneousData* scenario, the raw information is being read, and it is shown as “Heterogeneous Data Sources” data storage, and there are “Pollutants” and “Morbidity” data sources are created.

The second actor, named “Decision Maker”, is involved in an interactive process of decision making and choosing the optimal alternative. This actor communicates with agents by message passing through protocol ReturnSUI, stating the model, simulation values, prediction periods, levels of variable change, etc. It accepts the best alternative in accordance with its beliefs and the MAS. The flow of works, which are essential for decision making, include three scenarios: the Simulate models scenario, the Create recommendation scenario and the Search for the adequate model scenario; and three goals, which related to every scenario and have similar names. Each goal has a number of activities, and within each scenario are used, modified or created informational resources in form of data sources.

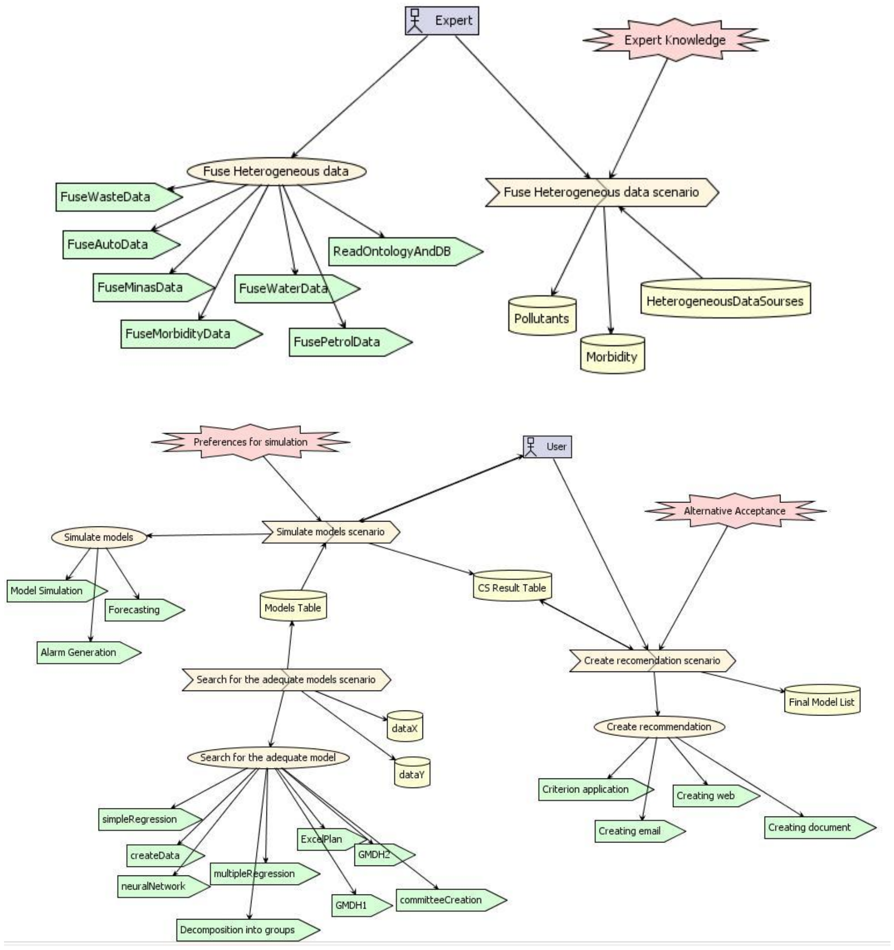

Fig. 5. The Prometheus diagram of MAS interaction with actors

In addition to the accepted MAS architecture and in order to gain time of the recommendation generation process and optimize interactions between agents, we used local agent teams, which coordinate and supervise task execution and resource usage. Agent teams have permitted to synchronize the work of the system, plans execution in a concurrent mode and strengthen the internal management by local decision making.

As we use four agent teams within the system: two within the first level, and one team on the second and third level, and each "main" agent plays several roles. In Table 1 we give a view of the correspondent logical levels, and the roles, which are played there. During the system work cycle, agents manipulate with diverse income and outcome information flows: data transmission protocols, messages, income and outcome data, etc. These information sources differ by the "life time":

Table 1. The roles played in the MAS

Logical level	Main agent	Subordinate agent	Role
Data Fusion	Data Aggregation agent	Domain Ontology agent Traffic Pollution Fusion agent Water Data Fusion agent Petroleum Data Fusion agent Mining Data Fusion agent Morbidity Data Fusion agent Waste Data Fusion agent	Data Fusion
	Data Pre-processing agent	Normalization agent Correlation agent Data Smoothing agent Gaps and Artifacts Check agent	Data Clearing
Data Mining	Function Approximation agent	Regression agent ANN agent GMDH agent Committee Machine agent Decomposition agent Evaluation agent	Impact Assessment Decomposition Function Approximation
Decision Making	Computer Simulation agent	Forecasting agent View agent Alarm agent	Computer Simulation Decision Making Data Distribution

they can be permanent and temporary, by the assessment levels – some can be used modified or deleted by agents, the decisions about others have to be taken by a system user. So, in this case the DPA has to operate as a planning agent, on the one hand, and has to pool the results of the subordinate agents' execution.

In the next section we shall describe the agents' organization in detail.

5 Description of the Agents within the MAS

5.1 The Data Aggregation Agent

The Data Aggregation agent (DAA) has a number of subordinate agents under its control; they are the Domain Ontology agent (DOA) and the *fusion agents*: the Water Data Fusion agent (WFA), the Petroleum Data Fusion agent (PFA), the Mining Data Fusion agent (MFA), the Traffic Pollution Fusion agent (TFA), the Waste Data Fusion agent (WDFA) and the Morbidity Data Fusion agent (MFA). First, the DAA sends the message *ReadOntology* to the DOA, which reads the OWL-file, which contains information about the ontology of domain, and makes it available to the DAA. The DOA terminates its execution, sending the message *OntologyIsBeingRead* to the DAA. Next, the DAA sends the message

Start Fusion to the fusion agents, which initiate their execution. When starting to execute, each fusion agent searches for the files that may contain information about the concept of its interest. Each fusion agent works with one or a few concepts of the domain ontology: WFA searches for the information about water contaminants and their properties, PFA – about the use of petroleum and related concepts, MDF retrieves data about the contamination related to mining industry activity, the WDFA retrieves data about wastes and its components, the TFA – data about transport vehicles activity, and the MFA – data about morbidity and their properties. When it finds the information file, the agent retrieves the information about the concept and its values, and changes their properties (in order to get rid of heterogeneity and to homogenize information) and sends it to the DAA, which pools retrieved information together. Finally, DAA fills the domain ontology with data, and puts data into a standard format. After that, the data files are ready to be pre-processed, and the DAA through the protocol ReturnDF tells the DPA that the data is fused and pre-processing can be started.

5.2 The Data Pre-processing Agent

The Data Pre-processing agent (DPA) provides data pre-processing and has a number of subordinate agents which specialize in different data clearing techniques: Normalization agent (NA), Correlation agent (CA), Data Smoothing agent (DSA), Gaps and Artifacts Check agent (GAA). They perform all data pre-processing procedures, including outliers and anomalies detection, dealing with missing values, smoothing, normalization, etc.

Fig. 6 gives a look at the first logical level, within which the Data Aggregation agent and the Data Pre-processing agent act. DPA starts to execute as soon as it receives a triggering message from DAA. The main function of the DPA is to coordinate the subordinate agents and decides when they are executed and in which order. Starting its execution, DPA sends the *StartDataConsistenceCheck* message, which triggers the GAA to eliminate artifacts, searches for double values and fills gaps. Having finished its execution, GAA sends to DPA a message. Then,

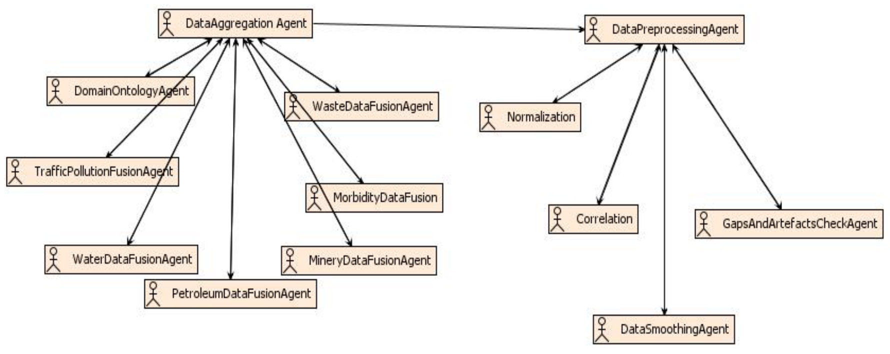

Fig. 6. Interaction between the Data Aggregation agent and the Data Pre-processing agent

DPA through the message *StartSmoothing* calls for DSA, which can execute exponential and weighted-average smoothing and terminates sending *SmoothingIsFinished* message to DPA. Then, NA and CA are called in their turn.

The outcomes of the DPA work are: data, ready for further processing and modeling, and additional data sources with correlation and normalization results.

5.3 The Function Approximation Agent

The Function Approximation agent (FAA) has a hierarchical team of subordinate agents, which serve to support the roles: "Impact Assessment", "Decomposition" and "Function Approximation" (see Fig. 7). FAA has under its control a number of *data mining agents*: the Regression agent (RA), the ANN agent (AA), and the GMDH agent (GMDHA), which work in a concurrent mode, reading income information and creating models. Then, if any agent from this group finishes modeling, it calls the Evaluation agent (EA), which evaluates the received models, and returns the list of the accepted ones. The others are banned and deleted. The FAA pools the outcome of the agents work, creates the list with the accepted models and then, once RA, AA and GMDHA finish their execution, calls the Committee Machine agent (CMA), which creates the final models in form of committees for each of the dependent variables, and saves them.

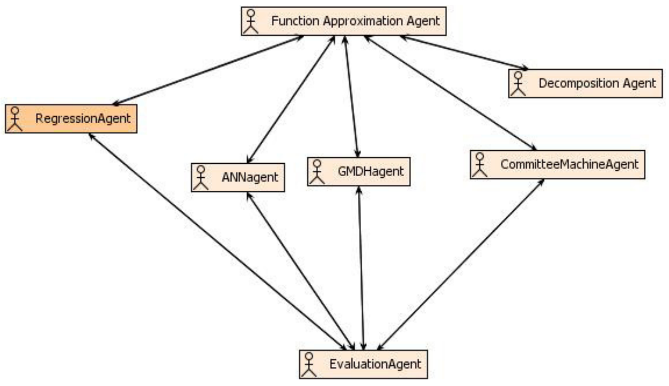

Fig. 7. Function Approximation agent and its team

The FAA working cycle is the following one. FAA sends *StartDecomposition* message and waits until DA finishes its execution. Then, having received the *StartDataMining* message, the data mining agents start execution in a concurrent mode. Each of them has plans with particular tools, and in case of AA, it has *neuralNetwork* and *evaluateImpactAssessment* plans, where the first plan is oriented to artificial neural network (ANN) creation and training, and the second plan aims to evaluate the environmental impact by means of ANN with determined structure and characteristics. EA is called by each of the *data mining* agents to evaluate the created models, and to check the adequacy of the model to the experimental data. EA is triggered by the *StartEvaluation* message from a *data mining* agent, and, whenever it is not busy, starts to execute. Having terminated the execution, it is ready to receive tasks and handle them. CM is the last to be called by FAA, as CM creates final hybrid models for every dependent variable. Each hybrid model is based on the previously created and evaluated models from the *data mining* agents, and uses the data sources created by them: *Models Table, IAResults*.

5.4 The Computer Simulation Agent

The Computer Simulation agent (CSA) interacts with the user and performs a set of tasks within Computer Simulation, Decision Making and Data Distribution roles. It has the agent team, which includes Forecasting agent (FA), Alarm agent (AmA) and ViewAgent (VA) as sown in Fig. 8.

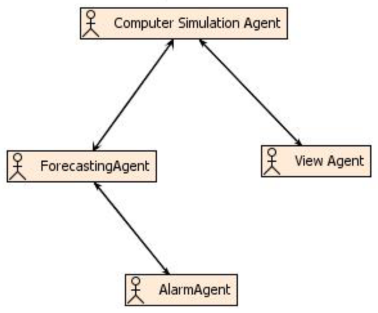

Fig. 8. Computer Simulation agent and its team

The CSA execution cycle starts by asking for the user preferences, to be more precise, for the information of the diseases and pollutants of interest, the period of the forecast, and the ranges of their value changes. Once the information from the user is received, CSA sends a message *SimulateAlternative* to FA, which reasons and executes one of the plans, which are Forecasting, ModelSimulation, and CriterionApplication. When the alternative is created, CSA sends the *StartAlarmCheck* message to AmA. The AmA compares the simulation and forecast data from the FA with the permitted and alarm levels for the correspondent indicators. If they exceed the levels, AmA generates alarm alerts.

6 Results

The MAS has an open agent-based architecture, which enables an easy incorporation of additional modules and tools, enlarging a number of functions of the system. The system belongs to the organizational type, where every agent obtains a class of tools and knows how and when to use them. Actually, such types of systems have a planning agent that plans the orders of the agents' executions. In our case, the main module of the Jack program carries out these functions. The ViewAgent displays the outputs of the system functionality and performs the interaction with the system user. As the system is autonomous and all the calculations are executed by it, the user has only access to the result outputs and the simulation window. He/she can review the results of impact assessment, modeling and forecasting and try to simulate tendencies by changing the values of the pollutants.

To evaluate the impact of environmental parameters upon human health in the Spanish region Castilla-La Mancha, in general, and in the city of Albacete in particular, we have collected retrospective data since year 1989, using open information resources offered by the Spanish Institute of Statistics and by the Institute of Statistics of Castilla-La Mancha. As indicators of human health and the influencing factors of environment, which can cause negative effect upon the noted above indicators of human health, the factors described in Table 2 were taken.

Table 2. Diseases and pollutants studied in research

Type of Disease/Pollutant	Disease class
Endogenous diseases:	Certain conditions originating in the prenatal period; Congenital malformations, deformations and chromosomal abnormalities.
Exogenous diseases:	Certain infectious and parasitic diseases; Neoplasm; Diseases of the blood and blood- forming organs and certain disorders involving the immune mechanism; Endocrine, nutritional and metabolic diseases; Mental and behavioral disorders; Diseases of the nervous system; Diseases of the eye and adnexa; Diseases of the ear and mastoid process; Diseases of the circulatory system; Diseases of the respiratory system; Diseases of the digestive system; Diseases of the skin and subcutaneous tissue; Diseases of the musculoskeletal system and connective tissue; Diseases of the genitourinary system; Pregnancy, childbirth and the puerperium; Symptoms, signs and abnormal clinical and laboratory findings, not elsewhere classified; External causes of morbidity and mortality.
Transport:	Number of Lorries, Buses, Autos, Tractors, Motorcycles, Others;

The MAS has recovered data from plain files, which contained the information about the factors of interest and pollutants, and fused in agreement with the ontology of the problem area. It has supposed some necessary changes of data properties (scalability, etc.) and their pre-processing. After these procedures, the number of pollutants valid for further processing has decreased from 65 to 52. This significant change was caused by many blanks related to several time series, as some factors have started to be registered recently. After considering this as an important drawback, it was not possible to include them into the analysis. The human health indicators, being more homogeneous, have been fused and cleared successfully.

The impact assessment has shown the dependencies between water characteristics and neoplasm, complications of pregnancy, childbirth and congenital malformations, deformations and chromosomal abnormalities. Part of Table 3 shows that within the most important factors apart from water pollutants, there are indicators of petroleum usage, mines outcome products and some types of wastes.

Table 3. Part of the table with the outputs of impact assessment

Disease Class	Pollutant, which influence upon the disease
Neoplasm	Nitrites in water; Miner products; DBO5; Dangerous chemical wastes; Fuel-oil; Petroleum liquid gases; Water: solids in suspension; Asphalts; Non-dangerous chemical wastes;
Diseases of the blood and blood- forming organs, the immune mechanism	DBO5; Miner products; Fuel-oil; Nitrites in water; Dangerous wastes of paper industry; Water: solids in suspension; Dangerous metallic wastes
Pregnancy, childbirth and the puerperium	Kerosene; Petroleum; Petroleum autos; Petroleum liquid gases; Gasohol; Fuel-oil; Asphalts; Water: DQO; DBO5; Solids in suspension; Nitrites.
Certain conditions originating in the prenatal period	Non-dangerous wastes: general wastes; mineral, constriction, textile, organic, metal. Dangerous oil wastes.
Congenital malformations, deformations and chromosomal abnormalities	Gasohol; Fuel-oil; DQO in water; Producing asphalts; Petroleum; Petroleum autos; Kerosene; Petroleum liquid gases; DBO5 in water; Solids in suspension and Nitrites.

The MAS has a wide range of methods and tools for modeling, including regression, neural networks, GMDH, and hybrid models. The function approximation agent selected the best models, which were: simple regression – 4381 models; multiple regression – 24 models; neural networks – 1329 models; GMDH – 2435 models. The selected models were included into the committee machines. We have forecasted diseases and pollutants values for the period of four years, with a six month step, and visualized their tendencies, which, in common, and in agreement with the created models, are going to overcome the critical levels. Control under the “significant” factors, which cause impact upon health indicators, could lead to decrease of some types of diseases.

As a result, the MAS provides all the necessary steps for standard decision making procedure by using intelligent computational agents. The levels of the system architecture, logically and functionally connected, have been presented. Real-time interaction with the user provides a range of possibilities in choosing one course of action from among several alternatives, which are generated by the system through guided data mining and computer simulation. The system is aimed to regular usage for adequate and effective management by responsible municipal and state government authorities.

We used as well traditional data mining techniques, as other hybrid and specific methods, with respect to data nature (incomplete data, short data sets, etc.). Combination of different tools enabled us to gain in quality and precision of the reached models, and, hence, in recommendations, which are based on these models. Received dependencies of interconnections and associations between the factors and dependent variables helps to correct recommendations and avoid errors.

To conclude, it is necessary to about our future plans regarding the work. As the work appeared to be very time consuming during the modeling, we are looking forward to both revise and improve the system and deepen our research. Third, we consider making more experiments varying the overall data structure and trying to apply the system to other but similar application fields.

7 Conclusions and Future Work

Agent-based decision making is a complicated problem, especially for a general issue as environmental impact upon human health. Though supposing it to be a tractable problem, we should note some essential advantages we have reached, and some directions for future research. Consequently, our future work can be drawn on various levels.

First, the MAS supports decision makers in choosing the behaviour line (set of actions) in general case, which is potentially difficult to analyse and foresee. As for any complex system, MAS allows pattern predictions, and the decision maker's choice is to be decisive. The framework we have created provides flows of works for decision generation, receiving raw data which are treated by agent teams, and transforming them into knowledge

Second, in spite of our time consuming modelling work, we are looking forward to both revise and improve the system and deepen our research. We are planning to refine data mining methods and modify some of them, to add some new ones, which can be especially valuable for concrete works, for example, some methods to work with qualitative information, fuzzy-based methods, dimension reduction methods, etc.

Third, as the system architecture has a general structure, we consider making more experiments varying data structure, and trying to apply the system to other application fields.

Acknowledgements

This work was partially supported by Spanish Ministerio de Ciencia e Innovación TIN2007-67586-C02 and TIN2010-20845-C03 grants, and by Junta de Comunidades de Castilla-La Mancha PII2I09-0069-0994 and PEII09-0054-9581 grants.

References

1. Sokolova, M., Fernández-Caballero, A.: A multi-agent architecture for environmental impact assessment: Information fusion, data mining and decision making. In: 9th International Conference on Enterprise Information Systems, ICEIS 2007, vol. AIDSS, pp. 219–224 (2007)
2. Chang, C.L.: A study of applying data mining to early intervention for developmentally-delayed children. Expert Systems with Applications 33(2), 407–412 (2006)
3. Gorodetsky, V., Karsaeyv, O., Samoilov, V.: Multi-agent and data mining technologies for situation assessment in security-related applications. Advances in Soft Computing, 411–422 (2005)

4. Sokolova, M., Fernández-Caballero, A.: Modeling and implementing an agent-based environmental health impact decision support system. Expert Systems with Applications 36(2), 2603–2614 (2009)
5. Bradshaw, J.M.: Software Agents. The MIT Press, Cambridge (1997)
6. Weiss, G.: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. The MIT Press, Cambridge (2000)
7. Bonczek, R.H., Holsapple, C.W., Whinston, A.B.: The evolving roles of models in decision support systems. Decision Sciences 11(2), 337–356 (1980)
8. Keen, P.G.W.: Adaptive design for decision support systems. ACM SIGMIS Database 12(1-2), 15–25 (1980)
9. Sprague, R.H., Carlson, E.D.: Building Effective Decision Support Systems. Prentice-Hall, Englewood Cliffs (1982)
10. Levin, M.S.: Composite Systems Decisions. Decision Engineering. Springer, Heidelberg (2006)
11. Power, D.J.: Decision support systems: concepts and resources for managers. Quorum Books, Westport (2002)
12. Chen, H., Bell, M.: Instrumented city database analysts using multi-agents. Transportation Research, Part C 10, 419–432 (2002)
13. Sokolova, M.V., Fernández-Caballero, A.: An agent-based decision support system for ecological-medical situation analysis. In: Mira, J., Álvarez, J.R. (eds.) IWINAC 2007. LNCS, vol. 4528, pp. 511–520. Springer, Heidelberg (2007)
14. Urbani, D., Delhom, M.: Water management policy selection using a decision support system based on a multi-agent system. In: Bandini, S., Manzoni, S. (eds.) AI*IA 2005. LNCS (LNAI), vol. 3673, pp. 466–469. Springer, Heidelberg (2005)
15. de Wolf, T., Holvoet, T.: Towards a full life-cycle methodology for engineering decentralised multi-agent systems. In: The Fourth International Workshop on Agent-Oriented Methodologies, pp. 1–12 (2005)
16. Vasconcelos, W.W., Robertson, D.S., Agusti, J., Sierra, C., Wooldridge, M., Parsons, S., Walton, C., Sabater, J.: A lifecycle for models of large multi-agent systems. In: Wooldridge, M.J., Weiß, G., Ciancarini, P. (eds.) AOSE 2001. LNCS, vol. 2222, pp. 297–318. Springer, Heidelberg (2001)
17. Bellifemine, F., Poggi, A., Rimassa, G.: Jade – A FIPA-compliant agent framework. Practical Applications of Intelligent Agents, 97–108 (1999)
18. Repast home page (2003), `http://repast.sourceforge.net`
19. Schelfthout, K., Holvoet, T.: ObjectPlaces: an environment for situated multi-agent systems. In: Third International Joint Conference on Autonomous Agents and Multi-agent Systems, pp. 1500–1501 (2004)
20. ISO/IEC 12207 home page, `http://www.iso.org/iso/`
21. Guarino, N., Giaretta, P.: Ontologies and knowledge bases: Towards a terminological clarification. In: Towards Very Large Knowledge Bases, pp. 25–32. IOS Press, Amsterdam (1995)
22. Samoylov, V., Gorodetsky, V.: Ontology issue in multi-agent distributed learning. In: Gorodetsky, V., Liu, J., Skormin, V.A. (eds.) AIS-ADM 2005. LNCS (LNAI), vol. 3505, pp. 215–230. Springer, Heidelberg (2005)
23. DeLoach, S.A., Wood, M.F., Sparkman, C.H.: Multiagent systems engineering. International Journal of Software Engineering and Knowledge Engineering 11, 231–258 (2001)

24. Wooldridge, M., Jennings, N.R., Kinny, D.: The Gaia methodology for agent-oriented analysis and design. Journal of Autonomous Agents and Multi-Agent Systems 3, 285–312 (2000)
25. Bauer, B., Müller, J.P., Odell, J.: Agent UML: a formalism for specifying multiagent software systems. International Journal of Software Engineering and Knowledge Engineering 11(3), 207–230 (2001)
26. Padgham, L., Winikoff, M.: Prometheus: A pragmatic methodology for engineering intelligent agents. In: Workshop on Agent Oriented Methodologies (Object-Oriented Programming, Systems, Languages, and Applications), pp. 97–108 (2002)
27. Giunchiglia, F., Mylopoulos, J., Perini, A.: The Tropos software development methodology: Processes, models and diagrams. In: Giunchiglia, F., Odell, J.J., Weiss, G. (eds.) AOSE 2002. LNCS, vol. 2585, pp. 162–173. Springer, Heidelberg (2002)
28. Gascueña, J.M., Fernández-Caballero, A.: Prometheus and INGENIAS agent methodologies: A complementary approach. In: Luck, M., Gomez-Sanz, J.J. (eds.) Agent-Oriented Software Engineering IX. LNCS, vol. 5386, pp. 131–144. Springer, Heidelberg (2009)
29. Bergenti, F., Gleizes, M.P., Zambonelli, F.: Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook. Springer, Heidelberg (2004)
30. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley & Sons, Chichester (2004)
31. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs (1995)
32. van Lamsweerde, A.: Goal-oriented requirements engineering: A guides tour. In: 5th IEEE International Symposium on Requirements Engineering, RE 2001, pp. 249–263 (2001)
33. Liu, L., Yu, E.: From requirements to architectural design: Using goals and scenarios. In: ICSE 2001 Workshop: From Software Requirements to Architectures, STRAW 2001, pp. 22–30 (2001)
34. JackTM Intelligent Agents home page, `http://www.agent-software.com/shared/home/`
35. Prometheus Design Tool home page, `http://www.cs.rmit.edu.au/agents/pdt/`
36. Gorodetsky, V., Karsaev, O., Konushy, V., Mirgaliev, A., Rodionov, I., Yustchenko, S.: MASDK software tool and technology supported. In: International Conference on Integration of Knowledge Intensive Multi-Agent Systems, pp. 528–533 (2005)
37. ISO 14031:1999. Environmental management - Environmental performance - Guidelines, `http://www.iso.org/`
38. International Classification of Diseases (ICD), `http://www.who.int/classifications/icd/en/`