In **Advanced Agent-Based Environmental Management Systems**,
Cortes, U. and Poch, M. (eds), Whitestein Series in Software Agent Technologies and Autonomic Computing,
Birkhauser, Springer, 2009, pp.119-138.

A methodology for developing environmental information systems with software agents

Ioannis N. Athanasiadis and Pericles A. Mitkas

Abstract. This chapter presents a unifying methodology for developing environmental information systems with software agents. Based on the experience reported in recent literature, we abstract common requirements of environmental information systems into agent types, combine state-of-the-art tools from computer science, service-oriented software engineering and artificial intelligence domains, as software agents and machine learning, and illustrate their potential for solving real-world problems. Specifically, two generic agent types are specified that behave as information carriers and decision makers, which provide an appropriate abstraction for deployment with added-value services in environmental management information systems.

A concrete pathway for applying these instruments throughout the software lifecycle of an environmental management information system is outlined, along with suggestions for software specification and deployment. The method is demonstrated in two application domains: one for air quality assessment and another for meteorological radar data surveillance.

Keywords. Intelligent information systems; Agent-oriented software engineering; Environmental data management, integration and reporting; Methodological tools; Agent architectures.

1. Introduction

1.1. Environmental Information Systems: Scope and challenges

Environmental Information Systems (EIS) is a broad term used for a range of IT systems related to natural resources data management. A working definition, given in [4] is the following one. *An Environmental Management Information System can be considered as an enterprise information system that provides efficient and accurate access to knowledge elements related to information about the natural environment.*

Environmental monitoring networks established worldwide, primarily in areas with potential pollution problems, observe and record the conditions of the natural environment. Through these networks, vast volumes of raw data are captured, and EIS are in charge of integrating all recorded data-streams. A typical EIS installation involves the fusion into a central database of environmental data recorded at distributed locations and in different means. Most commonly, EIS have been developed and installed to pursue one or more of the following goals:

a. *Off-line analysis systems.* Such systems are geared towards gathering historical data in a systematic way and making them available for in-depth analysis of natural phenomena.
b. *(Real-time) reporting systems.* These are systems responsible for identifying and reporting the current environmental conditions. They satisfy the public need for environmental awareness and the administrative and industrial needs for prevention measures.
c. *(Early) Warning Systems.* In this case, the goal is to prognosticate the future conditions of the environment. The need to forecast and forewarn about potential environmental problems is the key for preserving nature and taking precautionary actions.

Until lately, environmental data were meant for environmental scientists occupied with off-line studies and post-processing activities in their effort to understand the natural phenomena involved. However, there has been a transition in this practice: The aftermath of the growing societal interest in the environment and sustainable development was the emerging need for providing environmental information to the public.

Considering the quest for environmental information involving citizens, industry and administration, the challenge for EIS is to provide with *advanced information services*. EIS objectives are no longer restricted to integrate and process raw data-measurements. EIS are challenged to fuse information and diffuse knowledge, in forms comprehensible and accessible by everyone.

One challenge for modern EIS is to broaden their scope and embrace new users from the administration, industry, and the society. Modern EIS users have varying interpretations of environmental values, and consequently different needs in terms of detail of information and mediums for communication, but also conflicting interpretations of the data handled by EIS. In spite of their diverse needs, all users agree on the necessity to access trustworthy information *on time*. One of the major challenges for EIS today is to effectively capture, manage and report environmental information at "*near real-time*". Furthermore, modern EIS are called to develop personalized services, and to tackle issues related to data ownership and permissions, spatial and temporal scaling, industrial patent protection, intellectual property rights, and privacy issues.

Another challenge for EIS is that traditionally EIS are developed for certain case studies, therefore the generalization of the approach and the potential reuse of tools is a very seldom situation. This is partially an intrinsic characteristic of

environmental systems, as model configuration and adaptation to local conditions is required. Knowledge sharing, in any forms from raw data to sophisticated environmental model implementations, has become an increasingly important aspect of sound environmental management [4].

The above challenges can be met by adopting modular, service-oriented approaches, such as software agent technology, where reusable components can provide with added-value digital services in open environments.

1.2. Software agent technology

Agent-oriented software engineering has emerged as a novel paradigm for building software applications. The key abstraction used is that of an agent, as a software entity characterized by autonomy, reactivity, pro-activity, and social ability [12]. Certain types of software agents are able to infer rationally and support the decision making process [11]. Although there are variant definitions of the notion of agency found in literature (see the discussion in [23]), as a working definition we consider as a standard agent any computational system operating in some environment, that is capable for sensing its environment and act upon it in order to fulfill its goals ([22]). Agent-based systems may rely on a single agent, but the advantages of this initiative are revealed in the case of multi-agent systems, which consist of a community of co-operating agents. Several agents, structured in groups, can share perceptions and operate synergistically to achieve common system goals.

In agent-oriented software engineering, an agent is both a metaphor for software design and an abstraction for software development. As a software design metaphor, agents are considered as the building blocks of a system. Agent related technologies for software design include techniques for system requirements specification, software modelling, specification and verification (see discussion in [21]). Taking a step ahead, agent technology has moved to agent-oriented software engineering that adopts agents in the whole software design process, as for example in GAIA [26]. For agent-based development, there is a plethora of agent deployment strategies and toolkits (for an extended list the reader is directed to [13]), that vary from object-oriented programming and custom multi-agent systems to service oriented systems and agent platforms [14]. The latter have emerged as the evolution of object-oriented programming and distributed computing, and utilize agents as the basic software unit for developing software.

Agents are well-suited in open, competitive environments, as those of knowledge brokering, personal assistants, and online auctions, just to name a few. According to Parunak [17], software agents are best suited for applications that are modular, de-centralized, changeable, ill-structured, and complex. Parunak draws this conclusion, by ascertaining industrial and commercial applications, mainly in the fields of control systems, enterprise resource planning systems and electronic services.

2. Related work

EIS bear similar properties with the systems reviewed by Parunak: EIS need to address several users at different service levels, integrate data and information from heterogeneous sources, deal with data at multiple spatial and temporal scales and adapt to changing conditions. Also, they inherit both the uncertainty and the complexity involved in the natural phenomena. EIS involve uncertainties both at data, model and decision-making levels, and complexities related to the conflicting requirements and values of the involved users and stakeholders. Consequently, one could claim that the area of environmental informatics fits well with the competences of agent-based systems.

Agent technology has attracted a significant amount of attention from researchers in environmental informatics. Agent-based approaches have been adopted for developing environmental systems for data management, decision support or simulation purposes. Agent capabilities for distributed problem solving, adaptive personalized services, knowledge sharing and proactive autonomy may enable advanced digital services for EIS.

Since the '90s, agent technologies have been welcomed in ecological and environmental applications mainly as a metaphor for decomposing complex systems and studying the emergence of collective behavior [16]. Ever since, agent-based techniques have been extensively used for modeling in several environmental fields, including population dynamics, landscape modeling, water management, forest fires simulations, to name a few. However, agent technology has been adopted as a tool for software design and implementation of environmental applications in a limited, rather frag-mented way [1]. In a review of agent-based systems applied in environmental informatics [1], Athanasiadis studied twenty three systems that utilize agent technology at different stages of EIS development. An outlook of agent use in environmental software is illustrated graphically in Figure 1 (from [1]). The penetration of agent-oriented tools for software design and development is qualitatively represented on the two axes, and the acronyms of the systems reviewed is situated in the hyperplane.

While there are several systems that use agent modeling for system design, most of them have been implemented using traditional, object-oriented techniques. Implementations that employ agent platforms are still in infancy. Similar holds for agent oriented specification methodologies and tools that employ more sophisticated agent-oriented design toolkits. Only the PICO project [18] reports an agent-oriented software engineering technique throughout the whole design process, while software development using agent-based programming techniques is not accompanied with agent-based design to a great extend.

There is a lot of space for exploiting agent technology in EIS, by adopting agent-oriented software engineering and agent programming techniques in future developments. Also, one of the issues that have not been tackled so far, is a unifying methodology that abstracts common requirements of environmental information

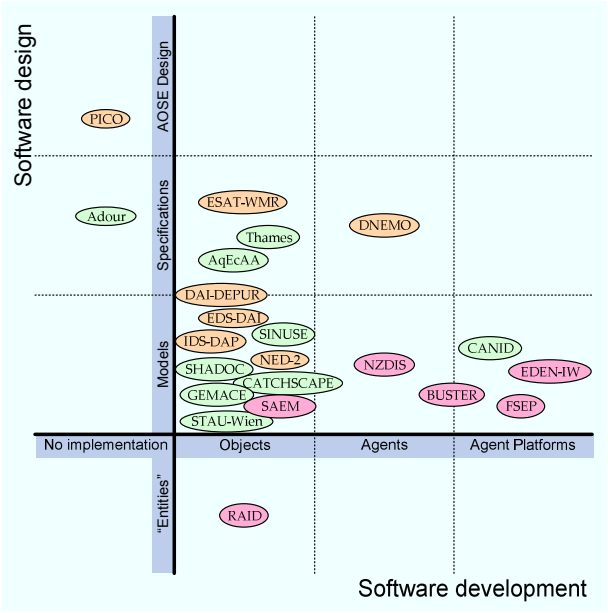

FIGURE 1. Penetration of agent-based techniques for software design and implementation in environmental informatics (from [1])

systems into agent types and provide an appropriate framework for deploying agent-based EIS.

3. Structural components of the methodology

3.1. Agent orientation in EIS lifecycle

Advancing on the way earlier research work has dealt with EIS using agent technology, we propose a methodology for developing EIS as a multi-agent systems (MAS). Our goal is to assign all services involved in the operational EIS to a software agent society. This work considers the notion of an **agent** as the basic building block for the requirements analysis, software specification, design and development for an EIS.

Agents are treated as *the* common conceptual tool that goes through the whole software development lifecycle of an EIS from its conception to final delivery and installation. Given the human-like characteristics of an agent, as their abilities to shape behaviors, realize roles and establish communicative dialogues, it makes them a very handy tool for modular, decentralized, intergrated software engineering process of an EIS. Agent orientation in EIS development can assist with tackling problems involved in environmental applications as domain complexity and interdisciplinarity. An agent in this way can be seen as a useful metaphor that

is much easier for wider audiences to comprehend, in contrast with the conventional software engineering paradigms. As a consequence, environmental scientists or the final users are enabled to follow easily the all software development phases of an EIS. Transparency in environmental software and accessibility to an extended peer community, involving other computer practitioners, disciplinary scientists, users and affected public, is considered [19] as a key requirement for increasing the trust against environmental software tools, and realizing the uncertainties involved. Agent orientation can provide natural solutions towards the direction of transparent, modular solutions.

Furthermore, software agent technology is a powerful tool for the development of advanced service development and provision in an EIS. Software agents are capable to formulate a mediating role, capable for providing services in open environments. The environmental information vacuum, underlined by Agenda 21, could be bridged using agents playing an information brokering role among diverse stakeholders and users. In this sense, the role of a software agent in environmental informatics is anticipated from a service oriented perspective. Software agents are service providers, mediating between end users and the environmental data pools for providing advanced information and decision support services.

We identify two main functions of software agents in EIS: software agents that behave as **information-carriers** or **decision-makers**. Agents as information carriers, act as a distributed community of data processing units, able to capture, manipulate and propagate information efficiently, i.e. they provide with data manipulation services. Agents as decision-makers, behave as a network of problem-solvers that work together to reach solutions.

In the followings we specify the behaviour of an agent in EMIS by defining a common external view, that defines how agents are interacting with their virtual environment, and two internal views: one for the information-carrier type and one for the decision-maker type. The internal views set out agent's own private behaviour and functionality.

3.2. An abstract agent for EIS external view

Based on the mediating profile of a software agent in environmental informatics, and using the notion of a standard agent [22], we define at an abstract level, an agent (*agent*) in EIS system as an autonomous entity that defines its actions based on its own perceptions about the state of its (virtual/artificial) environment.

Information on the state and the conditions of natural environment is captured in the form of **environmental data**. Environmental data may be a result of inspection, measurement, or simulation activities of scientists, and typically have spatio-temporal references.

Definition 3.1. Environmental data objects (EDO) constitute the virtual environment in which an *agent* operates. The (virtual/artificial) environment of an agent comprise all the possible states of the environmental data objects it percepts.

Let **O** be the set of environmental data objects that *agent* percepts. In an EIS each EDO can be considered as a function of space s and time t, therefore *agent* may be potentially exposed to a set of EDOs:

$$\mathbf{O} = \{O_{ED(1)}, O_{ED(2)}, \dots, O_{ED(i)}, \dots\}$$
$$\text{where } O_{ED(i)} = f(s,t)$$

The set **S** of the environmental states that an agent may percept is defined as the set of all instances of its environment. In principle, the states to which an agent is exposed to is an infinite set.

$$\mathbf{S} = \{s_1, s_2, \dots\}, \forall s_j = O_{ED(i)}(s,t)$$

Definition 3.2. Let $\mathbf{A} = \{a_1, a_2, \dots\}$ be the set of the possible actions of an *agent*, then the *agent* can be defined as a function that maps the sequences of the environmental states to agent actions as:

$$action : \mathbf{S}^\star \rightarrow \mathbf{A} \tag{3.1}$$

While the environment of the *agent* reacts to the the action $a \in \mathbf{A}$ applied on state $s \in \mathbf{S}$ as:

$$env : \mathbf{S} \times \mathbf{A} \rightarrow \mathcal{S} \tag{3.2}$$

Definition 3.3. The execution (run) of *agent* is the sequence:

$$run : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \dots \tag{3.3}$$

where $s_1 = env(s_0, a_0)$ is the state in which the agent environment goes when the action a_0 is performed on state s_0. The interaction between an agent and its environment is illustrated in Figure 2, that defines the external view of an *agent* of the toolbox.

Having defined the external view of a generic agent in an EIS, we proceed with the specification of the internal views of the two abstract agent types: the information carrier agent and the decision-maker agent.

3.3. Information-carrier agent internal view

The information-carrier agent role a_{IC} operates as a function that transforms EDO, in order to provide added-value data transfomation and dissemination services. From an external point of view, agent a_{IC} percepts a series of environmental states $\mathbf{S} = \{s_1, s_2, s_3, \dots\}$ to which it responds with a series of agent actions $\mathbf{A} = \{a_1, a_2, a_3, \dots, a_i\}$. Each action $a_i \in \mathbf{O}^\star$ i is an environmental data object that alters its evnironment's state.

Definition 3.4. The information carrier agent a_{IC} is an agent with state. The internal view on a_{IC} agent behaviour is illustrated in Figure 3. Each internal state **ik** is a set of EDO instances, therefore:

$$\mathbf{I} = \{i_1, i_2, i_3, \dots i_k\}, \qquad i_k \in \mathcal{P}(\mathbf{O}) \tag{3.4}$$

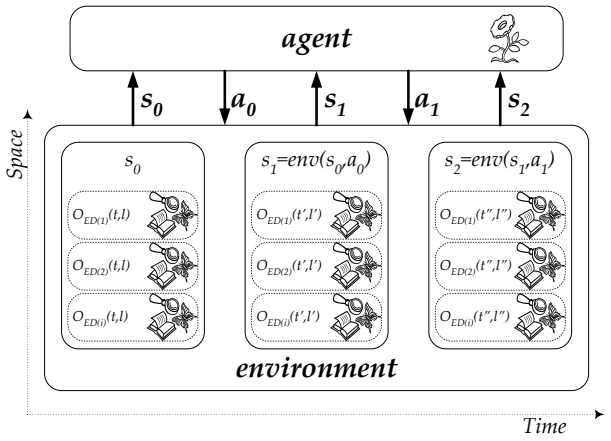

FIGURE 2. Software agent - virtual environment interaction in operational EMIS systems

The information carrier agent a_{IC} operates as follows: It observes its virtual environment through a perception function *see*, it captures the environmental states $s \in \mathbf{S}$ into agent perceptions $p \in \mathbf{P}$. Based on the sequences of perceptions, agent a_{IC} refreshes its internal state through the transformation function *trans*. Finally, a_{IC} performs its actions, based on its internal states via the *action* function.

Definition 3.5. The a_{IC} "internal state" can be specified as:

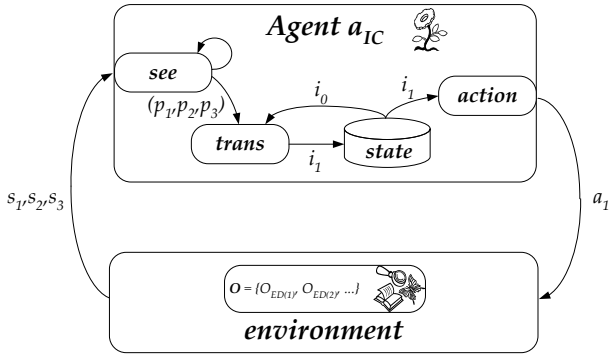

FIGURE 3. Internal structure and behaviour of agent a_{IC}

$$
\begin{aligned}
see &: \mathbf{S} \rightarrow \mathbf{P} && (3.5)\\
trans &: \mathbf{I} \times \mathbf{P}^{\star} \rightarrow \mathbf{I} && (3.6)\\
action &: \mathbf{I} \rightarrow \mathbf{A} && (3.7)\\
&\mathbf{P}, \mathbf{S} \subseteq \mathbf{OI} \subseteq \mathcal{P}(\mathbf{O})\mathbf{A} \subseteq \mathbf{O}^{\star}
\end{aligned}
$$

Implementing this behaviour an information carrier agent is capable of managing EDOs as follows: Suppose that initially a_{IC} enjoys the internal state i_0 and at some point it is exposed to three environmental states: s_1, s_2, s_3. Through the function *see* these environmental conditions are perceived by the a_{IC} as $p_1 = see(s_1), p_2 = see(s_2), p_3 = see(s_3)$. Then, the transformation function *trans* drives agent a_{IC} to the internal state $i_1 = trans((p_1, p_2, p_3), i_0)$, which causes it to return to its environment an action (sequence of EDOs) $a_1 = action(i(1)) = action(trans((p_1, p_2, p_3), i_0)$. This behaviour is illustrated in Figure 3.

The abstract behaviour specified, enables a_{IC} to perform a variety of data manipulation activities. From simple data capturing activites (i.e. from sensor networks) and database queries, to data aggregation and scaling, as well as complex transactions in an information system.

The set of all information carrier agents is noted by $\mathcal{IC}$ and contains all agents that implement the a_{IC} behaviour.

3.4. Decision-maker agent internal view

An agent a_{DM} functions as a decision maker by incorporating a reasoning engine *engine*, that implements a decision-making model. The decision making model can encompass deterministc strategies, knowledge-discovery techniques or heuristics.

Definition 3.6. The reasoning engine of a decision-maker agent a_{DM} is a mapping of internal states $i_1, i_2, \ldots i_n$ of a_{DM} to a decision d, following the relation:

$$engine : \mathbf{I}^{\star} \rightarrow \mathbf{D} \tag{3.8}$$

Agent a_{DM}, based on its decisions $d \in \mathbf{D}$ responds to the stimuli of its enviornment by performing a set of actions $a \in \mathbf{A}$.

Definition 3.7. Following the generic model of agent with state, agent a_{DM} operates as follows:

$$
\begin{aligned}
see &: \mathbf{S} \rightarrow \mathbf{P} && (3.9)\\
next &: \mathbf{I} \times \mathbf{P} \rightarrow \mathbf{I} && (3.10)\\
engine &: \mathbf{I}^{\star} \rightarrow \mathbf{D} && (3.11)\\
action &: \mathbf{D} \rightarrow \mathbf{A} && (3.12)\\
&\mathbf{P}, \mathbf{S} \subseteq \mathbf{OI} \subseteq \mathcal{P}(\mathbf{O})\mathbf{D} \subseteq \mathbf{OA} \subseteq \mathbf{O}^{\star}
\end{aligned}
$$

Let a_{DM} be in state i_0 and that at a certain point it is stimulated by observing an environmental state s_1. Through its *see* function it shapes the perception $p_1 = see(s_1)$, and consequently through function *next*, a_{DM} revises its internal state to $i_1 = next(i_0, p_1)$. Suppose that the state sequence (i_1, i_2) results the reasoning engine *engine* to get the decision $d_1 = engine(i_1, i_2)$. Due to this decision, a_{DM}, performs the action $a_1 = action(d_1) = action(engine(i_1, i_2))$. The internal state and the behaviour of an decision maker agent role are illustrated in Figure 4.

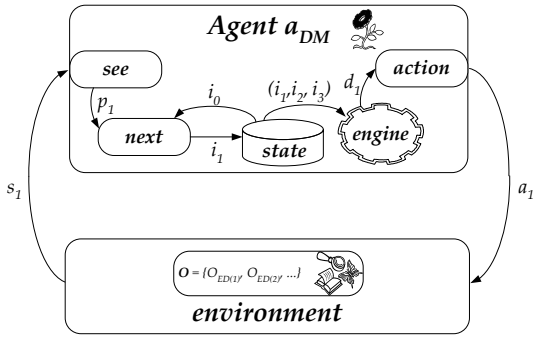

FIGURE 4. Internal structure and behaviour of agent a_{DM}

With the above internal model, we specify an agent with ability to infer, based on its virtual environment observations, to shape its perceptions and revise its internal state. Based on the sequence of its internal states it makes its decisions that feedbacks as actions to its environment. The set of all decision-maker agents is noted as $\mathcal{DM}$, that contains all agents that implement the behaviour of $a_{DM} \in \mathcal{DM}$.

4. Deployment of the toolbox

4.1. Multi-agent systems for EIS design and deployment

The advantages of agent technologies are revealed in the case of multi-agent systems, which consist of a community of co-operating agents. For the design and the development of EIS more than one agents of the two types defined above in our toolbox are cooperating together.

Definition 4.1. A multi-agent environmental information system model ($EnvMAS$) can be specified as a set of N agent types ag_n, each one of which implements the behavior of the information carrier agent type, or the decision-maker type.

$$EnvMAS_{model} = \{ag_1, ag_2, ag_3, \ldots, ag_n, \ldots, ag_N\} \qquad (4.1)$$
$$ag_n \in \{\mathcal{IC} \cup \mathcal{DM}\}, \qquad n = 1 \ldots N$$

Each agent type ag_n may have one or more instantiations $ag_n(k)$, $k = 1 \dots K_n$, and therefore the operating multi-agent system is specified as:

$$\begin{aligned} EnvMAS \quad = \quad & \{ag_1(1), ag_1(2), \dots ag_1(K_1), \\ & ag_2(1), ag_2(2), \dots ag_2(K_2), \\ & \dots, ag_n(1), ag_n(2), \dots, ag_n(K_n), \\ & \dots, ag_N(1), ag_N(2), \dots, ag_N(K_N)\} \end{aligned} \tag{4.2}$$

In total, there are $\sum_{n=1}^{N} K_n$ agents running in the system. $EnvMAS$ interacts with the natural environment and the end users through EDOs perceived by the agents, while inter-agent communication ensures system coordination.

4.2. How to get the toolbox to work

Having defined a generic agent for EIS, two specific agent types and a multi-agent system in our toolbox, the question that rises is how to use them in order to design and deploy an EIS as a multi-agent system. Though there are no silver bullets in software design, here we present a pathway that starts from an (unknown) application domain, decomposes it into agents of the toolbox, and synthesize them into an operating multi-agent system. An agent is a metaphor that is used throughout the software lifecycle. The specified abstract agent types are to be used for system analysis, software design and specification and software development.

4.2.1. System analysis phase. As in any software project, first comes the *system analysis phase*, which starts with a study of the domain at hand. Problem definition and the specification of the system goals is performed using abstract agent types as roles for defining system components and functions. This step can be realized by specifying the entities of the system using the toolbox. In principal, we identify two categories of entities:

(a) Entities that are part of the problem and contribute in the system specification;
(b) Entities that are part of the system and frame the problem boundaries.

In the first group fall all system drivers, which are external to the software system, as the end users. Such entities influence system specification, but they are not part of it. The second category comprise those entities which are part of the system. This role can be played by humans (i.e the system administrator), parts of the natural environment (i.e a river), or hardware or software components of the system (i.e a sensor network or a database system).

The analysis phase concludes with the specification of the entities, their behaviour and their interaction. This process can be done using requirement elicitation methods, as role playing games, where all stakeholders are involved. Environmental scientists, decision-makers, software architects and end users are engaged in this phase.

4.2.2. System design phase. Next comes the *system design phase*, which involves four main steps:

1. Entity behavior is assigned to software agent types;
2. The general system architecture is defined;
3. System functionalities are specified;
4. The system is specified using agent-based models.

The first step is to match system entities to certain agent types. The criterion for the assignment is the functionality and the behavior of entities. Information processing functionalities are assigned to information carrier agent types, and decision making nodes to decision maker agents. This step sketches a first draft of the system design, based on the requirements defined during the analysis phase.

Next, the general system architecture is specified, based on the functionality envisioned services. Agent behaviors are interwoven to ensure certain information flows through agent cooperation and coordination. Protocols for agent communication are identified and the external views of the agents are specified.

The third step is the functional specification of the system. Each if the agents is specified in detail and the internal views are detailed. Specifically, agent transformation functions are defined for information carrier agents. For decision-maker agents, internal states and inference engines are specified, as discussed below in paragraphs 4.2.3 and 4.2.4.

The design phase is concluded with agent oriented system design. Agent modeling toolkits, as GAIA [24, 26], is used for the detailed system specification, while the in-depth agent communication can be designed with AORML [20]. These above mentioned tools are a suggestion for specifying a multi-agent system as a computational organization; software architects may select alternative methods for agent-oriented system design, as AUML [15], iSTAR [25], or Tropos [10].

4.2.3. Information carrier agent design. Modeling agents as information carriers involves four steps:

Step 1. *Identify system inputs and outputs*:

Consider the interfaces between the software system, data sources and end-users. Identify services provided by system entities. Assign agents to realize these interfaces acting either as data fountains, or data sinks.

Step 2. *Formulate information channels*: Detail how information flows through the system. Specify possible data transformations needed. Assign those tasks to information carrier agents that operate as data managers.

Step 3. *Conceptualize agent messaging*: Based on the two previous steps, realize inter-agent communications for smooth information propagation. Specify the semantics of the communications using ontologies.

Step 4. *Specify delivery deadlines*: Concrete deadlines are assigned to agent communication, in order to ensure 'on-time' delivery of information. Exit on failure strategies need to be detailed too.

The outcome of the above procedure is materialized as the specifications of a MAS architecture, in the form of:

$$\text{MAS} = \langle A, O, I, D \rangle \tag{4.3}$$

where:

- $A = \{\text{ag}_1, \ldots, \text{ag}_n\}$, is a countable set of software agents.
- O is the domain ontology, which specifies the common vocabulary in order to represent the system environment.
- $I = \{\text{I}_k = (\text{ag}_i, \text{ag}_j)/\text{ag}_i, \text{ag}_j \in A\}$, is a set of interactions between agents. These interactions show the relations in the system organization and they allow the definition of a social framework determining the information flows in the system.
- $D = \{\text{D}_k, \forall\ \text{I}_k \in I\}$, is a set of the delivery deadlines assigned to each agent communication.

4.2.4. Decision-maker agent design. Agents as decision makers are employed to deliver the reasoning abilities of the EIS. Indicatively, decision-making in a real-time EIS involves either assessment services or activities to overcome data uncertainty problems. Based on the domain knowledge, agent decision-making strategies are identified through the following procedure:

Step 1. *Problem formulation and decomposition*: Consider the overall problem at hand and try to break it down into sub-problems.

Step 2. *Construction of decision points*: Assign specific agents to solve each sub-problem, taking under account their resources, specified by the system's architecture.

Step 3. *Decision strategy specification*: For each sub-problem provide a strategy to solve it using the available resources.

Step 4. *Realization of Inference models*: Implement the decision strategies designed in the previous step as inference models of the respective agents. Inference models will be embedded into decision-maker agents as reasoning engines.

This procedure is highly dependent on the application under consideration. Finding an optimal decision strategy is a rather difficult task, especially when execution time is a parameter of success. However, three distinct cases of decision-making engines, can be identified, covering the majority of applications:

Case 1 *Deterministic Strategies*: These are applied, when domain-specific, certain, explainable rules for decision-making are available. Such rules may encompass natural laws, logical rules or legal constraints. In such cases, rules are incorporated as a static, confident, explainable expert system into the agents.

Case 2 *Data-driven Strategies*: When historical datasets are available, the application of machine learning algorithms for knowledge discovery can yield interesting knowledge models. These models can be used for agent reasoning in a dynamic, inductive way. In EIS, there are large volumes of data continually recorded. When natural laws describing the monitored phenomena do not exist, or they are too complex, data-driven models, such as decision trees, case-based reasoning, or neural networks provide an option to the application developer. In

this case, the procedure involves the creation of an inference model from historical data. This model is later incorporated into the agents.

Case 3 *Heuristic strategies*: When neither of the above cases is applicable, heuristic models or 'rules of thumb' may be incorporated into agents.

This checklist provides with a guideline for designing decision-making agents required by a multi-agent EIS.

4.2.5. System development and deployment phase. *System development* is the third phase of the process. Having specified the overall agent architecture, the internal agent structures and agent communication protocols in the previous step, next comes the system deployment using an agent platforms. Software engineer has a plethora of tools available for agent programming and deployment, as JADE [9, 8]. Agent programming toolkits consist a middleware for the development of distributed multi-agent applications, that support natively peer-to-peer agent communication, basic agent behaviors and an agent runtime environment.

The development phase concludes with system installation and deployment. An iterative process for revealing design faults or development bugs is then required for ensuring software quality of the system's final version.

5. Demonstration of the methodology

The methodology presented here has been demonstrated in two case studies which we discuss below.

5.1. Application in air quality assessment

The methodology described in the previous section has been applied to the development of O_3RTAA, an EIS for air quality assessment and reporting. O_3RTAA is a multi-agent system for monitoring and assessing air quality, by exploiting data from a sensor network. A community of software agents is assigned to monitor and validate measurements coming from several sensors, to assess air-quality, and, finally, to fire alarms to appropriate recipients, when needed, via the Internet. The overall system architecture is depicted in 5.

In O_3RTAA, information carriers are responsible to collect data from field sensors, perform data management activities, as data preprocessing, normalization and transformation, and propagate information, which involves posting information to end-users over the internet and updating a measurement database. As shown in Figure 5, "contribution agents" (CA) operate as data fountains of the system, which capture data from the sensors and "distribution agents" (DA) are data sinks which provide with information services to end users.

Decision-maker agents in O_3RTAAare responsible for validating incoming measurements; substituting erroneous measurements by estimating missing values and approximating false sensor readings; and calculating of qualitative indicators. The first two activities are left to CA agents, while "Alarm DMA" agent is in charge of the third. Data-driven strategies are employed for data validation and

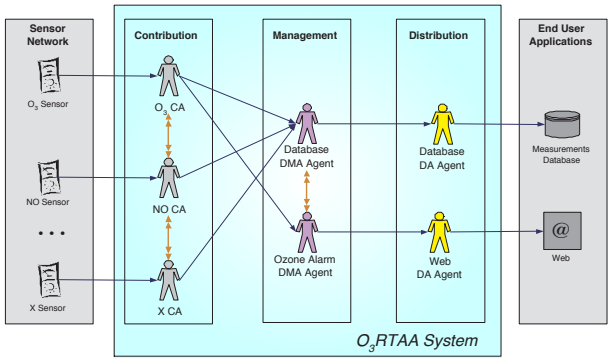

FIGURE 5. O_3RTAA System Architecture

erroneous data substitution engines, while deterministic strategies were used for for air quality indicator engine.

The O_3RTAA system has been tested against real data and demonstrated as a pilot case at the Mediterranean Centre for Environmental Studies Foundation (CEAM), in collaboration with IDI-EIKON, Valencia, Spain. More details on O_3RTAA architecture are given in [6] and a more generalized framework is presented in [2]. Data-driven strategies using knowledge discovery techniques are presented in [7, 3]

5.2. Application in meteorological radar data surveillance

The second case study is a meteorological radar data surveillance system deployed as a pilot service for the Meteorological Service of Cyprus. The agent-based EIS developed, called ABACUS intervenes between a meteorological Doppler radar and end-users, as the meteorological service and the local airport. The goal of the system is to manage and process radar recordings (which indicate clouds formations above the island); identify specific meteorological incidents and their evolution through time, and to provide with digital services to the end users, as online warnings and visualizations.

This goal was assigned to a community of cooperating agents, illustrated in Figure 6. An information carrier agent is responsible for acquiring radar scans and preprocess them by applying certain filter. A set of meteorologist agents (decision-maker agents), each one of which is responsible for an annular sector within the radars range, calculates metrics and indices within its sector and applies decision rules for assessing the weather conditions and issuing alarms. Finally, a couple of information carrier agents further process the data and presents them to the end users.

ABACUS has been demonstrated with real data at the Meteorological Service of Cyprus. System architecture is detailed in [5].

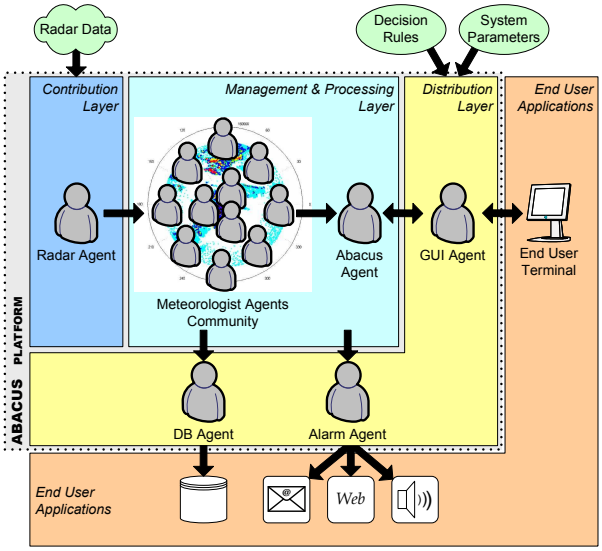

FIGURE 6. Abacus system architecture

6. Discussion

In this work, we presented an methodology for developing agent-based EIS, that supply with digital services, as those of seamless data integration, environmental assessment, warning services, information diffusion, and advanced decision making. The method relies on two generic agent types for EIS and a concrete pathway for using agents as a unique metaphor for system analysis, design and deployment.

An overview of the methodology is depicted in Figure 7. It unifies in a single process two properties of agents: their capacity (a) for distributed information processing, and (b) for distributed problem solving. The main advantage of such an approach is that it fully exploits the capabilities of autonomous agents, considering them as both information carriers and decision makers. Information flows dictate how agents manipulate data, while domain knowledge determines the decision making process incorporated into the agents. Information flows are implemented through agent communication channels, while the decision-making processes are transformed into agent reasoning models. Agent architectures that can be designed with our methodology are able to deal with data uncertainty problems, through the hybrid use of either deterministic, data-driven or heuristic decision-making strategies for agent reasoning.

Our methodology provides with the means for adopting agent technology throughout the lifecycle of environmental information systems. We defined where

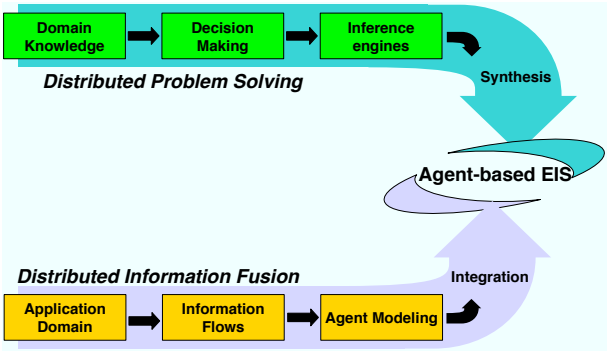

FIGURE 7. An abstract view of our methodology

and how agents can be used, suggested existing tools from agent oriented software engineering and provided with guidelines about the process that a software architect has for developing agent-based EIS. The

The benefits of our methodology rely on two pillars: First is the use of agents for software requirement analysis and design, as the human like characteristics of agents are much easier for the environmental scientist to comprehend and communicate with. This means that environmental scientists can be engaged deeper in the EIS development process. Second, it employs a distributed information processing approach, using software agents, thus agent-based EIS are open, modular and extensible, which is always a goal for EIS. Finally, we argued that using a single metaphor (that of an agent) throughout the EIS lifecycle is a great advantage for building trust of environmental scientists in EIS.

References

[1] I. N. Athanasiadis. A review of agent-based systems applied in environmental informatics. In A. Zerger and R. M. Argent, editors, *MODSIM 2005 Int'l Congress on Modelling and Simulation*, pages 1574–1580, Melbourne, Australia, December 2005. Modelling and Simulation Society of Australia and New Zealand.

[2] I. N. Athanasiadis. An intelligent service layer upgrades environmental information management. *IT Professional*, 8(3):34–39, May-June 2006.

[3] I. N. Athanasiadis. The Fuzzy Lattice Reasoning Classifier for mining environmental data. In V. G. Kaburlasos and G. X. Ritter, editors, *Computational Intelligence Based on Lattice Theory*, Studies in Computational Intelligence, pages 175–193. Springer-Verlag, 2007.

[4] I. N. Athanasiadis. Towards a virtual enterprise architecture for the environmental sector. In N. Protogeros, editor, *Agent and Web Service Technologies in Virtual Enterprises*, pages 256–266. Information Science Reference, 2007.

[5] I. N. Athanasiadis, M. Milis, P. A. Mitkas, and S. C. Michaelides. A multi-agent system for meteorological radar data management and decision support. Under revision, 2008.

[6] I. N. Athanasiadis and P. A. Mitkas. An agent-based intelligent environmental monitoring system. *Management of Environmental Quality*, 15(3):238–249, 2004.

[7] I. N. Athanasiadis and P. A. Mitkas. Knowledge discovery for operational decision support in air quality management. *Journal of Environmental Informatics*, 9(2):100–107, Jul 2007.

[8] F. Bellifemine, G. Caire, A. Poggi, and G. Rimassa. JADE-A white paper. *EXP in search of innovation*, 3(3):6–19, September 2003.

[9] F. Bellifemine, A. Poggi, and G. Rimassa. Developing multi-agent systems with JADE. In *Proc. of 7th Int'l Workshop on Agent Theories, Architectures and Languages (ATAL-2000)*, Boston, MA, 2000. Available online:http://jade.cselt.it.

[10] F. Giunchiglia, J. Mylopoulos, and A. Perini. The Tropos software development methodology: processes, models and diagrams. In F. Giunchiglia, J. Odell, and G. Weiss, editors, *Software Engineering III, Third International Workshop, AOSE-2002*, LNCS. Springer-Verlag, 2003.

[11] N. Jennings, K. Sycara, and M. J. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[12] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[13] M. Luck, P. McBurney, and C. Preist, editors. *Agent Technology: Enabling Next Generation Computing, A Roadmap for Agent Based Computing*. AgentLink, 2003.

[14] E. Mangina. *Review of software products for Multi-Agent Systems*. AgentLink, 2002.

[15] J. Odell, H. v. D. Parunak, and B. Bauer. Extending UML for agents. In *Proc. of the 2nd Int'l Workshop on Agent-Oriented Information Systems*, Berlin, Germany, 2000. iCue Publishing.

[16] R. L. Olson and R. A. Sequeira. An emergent computational approach to the study of ecosystem dynamics. *Ecological Modelling*, 79:95–120, 1995.

[17] H. v. D. Parunak. Agents in Overalls: Experiences and Issues in the Development and Deployment of Industrial Agent-Based Systems. *International Journal of Cooperative Information Systems*, 9:209–227, 2000.

[18] A. Perini and A. Susi. Developing a decision support system for integrated production in agriculture. *Environmental Modelling & Software*, 19:821–829, 2004.

[19] J. Rotmans. Methods for Integrated Assessment: The challenges and opportunities ahead. *Environmental Modeling and Assessment*, 3:155–179, 1998.

[20] G. Wagner. The Agent-Object-Relationship metamodel: Towards a unified conceptual view of state and behavior. *Information Systems*, 28(5):475–504, 2003.

[21] G. Weiss. Agent orientation in software engineering. *Knowledge Engineering Review*, 16(4):349–373, 2002.

[22] M. Wooldridge. Intelligent Agents. In G. Weiss, editor, *Multiagent Systems: A modern approach to distributed Artificial Intelligence*, chapter 1, pages 27–78. MIT Press, 2000.

[23] M. Wooldridge and N. R. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[24] M. Wooldridge, N. R. Jennings, and D. Kinny. The GAIA Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[25] E. Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Proc. of the 3rd IEEE Int. Symp. on Requirements Engineering*, Washington, USA, 1997. IEEE.

[26] F. Zambonelli, N. R. Jennings, and M. Wooldridge. Developing multiagent systems: the GAIA Methodology. *ACM Trans on Software Engineering and Methodology*, 12(3):317–370, 2003.

Ioannis N. Athanasiadis
Dalle Molle Institute for Artificial Intelligence,
Lugano, Switzerland
e-mail: `ioannis@athanasiadis.info`

Pericles A. Mitkas
Electrical and Computer Engineering Dept, Aristotle University of Thessaloniki,
Thessaloniki, Greece
e-mail: `mitkas@eng.auth.gr`