Volume 28 Number 1 April 2004 ISSN 0350-5596

Informatica

An International Journal of Computing and Informatics

Special Issue:

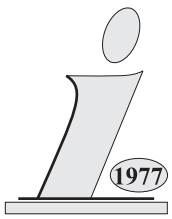
Agent Based Computing

Guest Editor:

Gabriel Ciobanu

Marcin Paprzycki

Shahram Rahimi



The Slovene Society Informatika, Ljubljana, Slovenia

EDITORIAL BOARDS, PUBLISHING COUNCIL

Informatica is a journal primarily covering the European computer science and informatics community; scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor from the Editorial Board can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the list of referees. Each paper bears the name of the editor who appointed the referees. Each editor can propose new members for the Editorial Board or referees. Editors and referees inactive for a longer period can be automatically replaced. Changes in the Editorial Board are confirmed by the Executive Editors.

The coordination necessary is made through the Executive Editors who examine the reviews, sort the accepted articles and maintain appropriate international distribution. The Executive Board is appointed by the Society Informatika. Informatica is partially supported by the Slovenian Ministry of Science and Technology.

Each author is guaranteed to receive the reviews of his article. When accepted, publication in Informatica is guaranteed in less than one year after the Executive Editors receive the corrected version of the article.

Introduction

Recent years can be characterized by a constantly increasing interest in software agents and agent systems. In order to increase the number of avenues through which researchers working in these areas can exchange ideas, we have proposed a special session devoted to Agent Based Computing (ABC), which took place during the *7th SCI Conference* in Orlando, Florida in July, 2003. Our initiative was successful. We have received 18 submissions, out of which, after careful refereeing, we have selected 9 which were then presented and published in conference proceedings. Unfortunately, the length of papers allowed to be published was limited to 6 pages. Since, realistically speaking, it is difficult to present a complete picture of ones work on 6 pages, we have started to look for a journal to publish full papers. Thanks to professor Matjaz Gams, we have found it in the *Informatica* journal, and we would like to use this occasion to express our gratitude to Professor Gams.

This *Special Issue* consists of nine papers. The initial four are devoted to lower level functionalities and tasks involved in agent system development. First, *Representing agents and their systems: a challenge for current modeling languages* by Renato Levy and James Odell, argues that while many of the evolutionary aspects of agent modelling can accomplished by extending current modelling languages such as UML 2.0; the revolutionary aspects, however, will probably require new approaches. In the second, *An XML-based serialization of information exchanged by software agents*, Sînică Alboaie, Sabin Buraga and Lenuța Alboaie propose an XML-based model that can be used in serialization of objects processed by mobile agents. The third paper, *A task-oriented compositional mobile agent architecture for knowledge exchanges between agencies and agents*, by Hong Zhou, Shahram Rahimi, Yufang Wang, Dia Ali and Maria Cobb discusses how task-oriented compositional structure allows assembly of new mobile agents from existing components. Finally, in *Towards a modeling methodology for fault-tolerant multi-agent systems*, Sehl Mellouli, Bernard Moulin and Guy W. Mineau present an overview of agent oriented software engineering methodologies, and propose some guidelines towards a modeling methodology for fault-tolerant MAS.

The remaining five papers involve practical applications of agent technology. In *System administration using software agents*, Shahram Rahimi and Santosh Ramakrishna introduce ABSA; an agent-based solution to automated system administration that supports multiple system administration features such as domain-wise administration, automated error handling and default system configuration. *Collaborative translation with mobile agents*, by Eric Sanchis, Jean-Louis Selves and Zhao Yang Pan proposes a lightweight peer to peer architecture based on mobile agents which implement a model of mobility called actual mobility, applied to collaborative translation. In *Human-agent interaction: case studies in human supervised UAV*, Henry Hexmoor and Srinivas Battula offer case studies of empowering agents with adjustment of cognitive notions of autonomy and trust that enable them to have social abilities in interacting with a human supervisor. Their application domain is control of unmanned aerial vehicles. Paper by Marcin Paprzycki, Austin Gilbert, Andy Nauli, Minor Gordon, Steve Williams, and Jimmy Wright, entitled *Indexing agent for data gathering in an e-travel system,* discusses the problem of indexing information available on the Internet with the ultimate goal of delivering personalized content to users of an agent-based travel support system. Finally, in *Multi-agent system case studies in command and control, information fusion and data management,* Frederick Sheldon, Thomas Potok and Krishna Kavi discuss three different agent-based development projects: (1) distributed command and control (DCC) in fault-tolerant, safety-critical responsive decision networks, (2) agents discovering knowledge in an open and changing environment, and (3) lightweight distributed data management (DM) for analyzing massive scientific data sets. These case studies are used to characterize the fundamental commonalities and benefits of agent based systems in light of experiences gathered during their deployment.

We would like to thank our referees: Dia Ali, Giacomo Cabri, Maria Cobb, Sabin Corneliu Buraga, Stan Franklin, Violetta Galant, Henry Hexmoor, Sofiane Labidi, Vincenzo Loia, Armin R. Mikler, Fred Petry, Tomas E. Potok, Stanisław Stanek, Ron Sun, and Tatyana Yakhno for their diligent work. Without their effort we would not be able to complete this special issue.

Gabriel Ciobanu, Marcin Paprzycki and Shahram Rahimi

Representing Agents and their Systems: A Challenge for Current Modeling Languages

Renato Levy
Intelligent Automation, Inc.
7519 Standish Place, Suite 200, Rockville, MD 20855 USA
rlevy@i-a-i.com

James Odell
James Odell Associates
3646 W. Huron River Drive
Ann Arbor, MI 48103 USA
email@jamesodell.com

Keywords: Systemic, Agents, Multi-agent systems, UML, AUML, model, modeling languages, modeling notation

Received: July 15, 2003

Leading-edge organizations are now developing systems that employ autonomous, interactive entities, or agents. [1; 2] Compared to its predecessors, the agent-based approach is evolutionary. However, its usages could be revolutionary. This paper begins by presenting some of the differences and similarities between agents and previous approaches. We then discuss some of the challenges for using current modeling approaches to represent agent-based systems. Our position is two folded: many of the evolutionary aspects of agent modeling can accomplished by extending current modeling languages such as UML 2.0; while the revolutionary aspects, however, will probably require new approaches.

1 Introduction

Advances on technology and on system's theory (non-linearity, complexity and chaos theory) has led to engineers to challenge problems which had been deemed intractable for a number of years. These problems are usually NP-hard in high order, which makes even the development of efficient heuristics a very complex challenge. Observation of how nature deals with problems of such complexity led to a different approach to software development, known as agent-based software, which has been successful in developing solutions for such problems. The agent-based software paradigm has established itself as viable approach for developing software directed towards control and simulation of complex systems.

Figure 1 illustrates one way of thinking about the evolution of programming paradigms. Originally, the basic unit of software was the complete program where the programmer had full control. The program's state was the responsibility of the programmer and its invocation determined by the system operator. The term modular did not apply because the behavior could not be invoked as a reusable unit in a variety of circumstances.

As programs became more complex and memory space became larger, programmers needed to introduce some degree of organization to their code. The modular programming approach employed smaller units of code that could be reused under a variety of situations. Here, structured loops and subroutines were designed to have a high degree of local integrity. While each subroutine's code was encapsulated, its state was determined by

	Monolithic Programming	Modular Programming	Object-Oriented Programming	Agent Programming
Unit Behavior	Nonmodular	Modular	Modular	Modular
Unit State	External	External	Internal	Internal
Unit Invocation	External	External (CALLed)	External (message)	Internal (rules, goals)

Figure 1: Evolution of programming approaches [3].

externally supplied arguments and it gained control only when invoked externally by a CALL statement. This was the era of procedures as the primary unit of decomposition.

In contrast, object orientation added to the modular approach by maintaining its segments of code (or methods) as well as by gaining local control over the variables manipulated by its methods. However in traditional OO, objects are considered passive because their methods are invoked only when some external entity sends them a message.

Software agents have their own logical thread of control, localizing not only code and state but their invocation as well. Such agents can also have individual rules and goals, making them appear like "active objects

with initiative." In other words, when and how an agent acts is determined by the agent.

At each evolutionary step, then, various modeling languages were created to aid system developers. The latest and most popular graphical language is the Unified Modeling Language (UML) developed by the Object Management Group (OMG). As agent based systems starts their transition from university and research labs into mainstream engineering, grows the necessity for appropriate graphical languages and tools to support it. Since agent technology can be viewed as an evolution on previous technologies, it would be reasonable to believe that agent-based languages can be based on previous approaches — at least in part. However, the way in which agents can be used for application systems is far richer than earlier approaches. Here, we may also need to develop new languages to accommodate the agent-based approach, in addition to adopting and modifying pre-agent languages.

The rest of this paper is organized as follows: In section 2 we present the philosophical differences between agent systems and their predecessor software engineering paradigms. Section 3 demonstrates how these philosophical differences impact our ability to represent such systems in current modeling languages, and specifically in UML. In section 4, we proposed a set of alternative representations that are able to solve some of the previous modeling limitations and in section 5 we present a study case in which some of the challenges and proposed solutions are debated. Section 6 concludes this paper with an invitation for an open debate about the issues raised.

2 Philosophical Differences

Agents are commonly regarded as autonomous entities, because they can watch out for their own set of internal responsibilities. Furthermore, agents are interactive entities that are capable of using rich forms of messages. These messages can support method invocation—as well as informing the agents of particular events, asking something of the agent, or receiving a response to an earlier query. Lastly, because agents are autonomous they can initiate interaction and respond to a message in any way they choose. In other words, agents can be thought of as objects that can say "No"—as well as "Go." Due to the interactive and autonomous nature of agents, little or no iteration is required to physically launch an application. Van Parunak summarizes it well: "In the ultimate agent vision, the application developer simply identifies the agents desired in the final application, and the agents organize themselves to perform the required functionality." [3] No centralized thread or top-down organization is necessary since agent systems can organize themselves.

However, several other key areas exist that differentiate the agent-based approach from traditional approaches such as OO. The list below describes some underlying concepts that agent-based systems can employ. None are universally used by agents: active object systems may use them as well. Furthermore, no agent system is required to use all of them. This list merely provides a "menu" of features that agent systems can —and often do — employ.

Decentralization: Objects can be thought of as centrally organized, because an object's methods are invoked under the control of other components in the system. Yet, some situations require techniques that are decentralized and self-organized. For example, classical ballet requires a high degree of centralization called choreography, while at the other extreme the processes of nature involve a high degree of individual direction. However, most businesses require a balance of standardized procedures and individual initiative: one extreme or the other would be detrimental to the business.

Supply-chain systems can be planned and centrally organized when the business is basically stable and predictable. In unstable and unpredictable environments, supply chains should be decentralized and self-organized (an option not supported by commercial supply-chain systems today). Agent-based environments can employ both centralized and decentralized processing. While agents can certainly support centralized systems, they can also provide us with the ultimate in distributed computing.

Multiple and dynamic classification: In OO languages, objects are created by a class and, once created, may never change their class or become instances of multiple classes (except by inheritance). Agents can provide a more flexible approach. For example, a particular agent can be a person, employee, spouse, landowner, customer, and seller all at the same time or at different times. When the agent is an employee, that agent has all the state and procedural elements consistent with being an employee. If the agent is terminated from his or her job, the employment-related state and procedural elements are now longer available to the agent. Whether employed or not, the agent is still the same entity—it just has a different set of features. The ability to express roles and role changes is not new to OO. However, most OO languages do not directly support this mechanism (even though UML does).

Furthermore, agents might play different roles in different domains. When you go to work, you play the employee role. When you return home, you change roles—for example, playing the spouse role. OO languages do not directly support such domain-dependent mechanisms that are necessary for agent-based environments. The single-class OO approach is efficient and reliable; the multiple and dynamic approach provides flexibility and more closely models our perception of the world. Agents can use either approach; the choice belongs to the system designer.

Instance-level features: The features possessed by each object are defined by the object's class—a benefit enjoyed by agents as well. However, each agent may also acquire or modify its own features, i.e., features that are not defined at the class level, but at the individual agent (or instance) level. In other words, if an individual agent has the ability to learn, it can change its own behavior— permitting it to act differently that any other

agent. If an agent can change itself, it can add (as well as subtract) features dynamically. For example, with genetic programming software, agents are created genetically. Here, each parent contributes some portion of an offspring agent's genetic string—much in the same way that occurs in nature. This approach is particularly popular in one area of agent-based systems known as artificial life. (Artificial life is the study of man-made systems that exhibit the behavioral characteristic of natural living systems. It models life-as-we-know-it within the larger picture of life-as-it-should-be.)

Emergence: The interaction of many individual agents can give rise to secondary effects where groups of agents behave as a single entity. For example, ant colonies, flocks of birds, and stock markets have emergent qualities. Each consists of individual agents acting according to their own rules and even cooperating to some extent. Yet, ants colonies thrive, birds flock, and markets achieve global allocations of resources—all without a central cause or an overall plan. Agents can possess just a few very simple rules to produce emergence. In fact, when constructing agent-based systems, starting out with simple agents is important, because emergence is then easier to understand and harness. More complexity can be added over time to avoid being overwhelmed.

Since traditional objects do not interact without a higher level thread of control, emergence does not usually occur. As more agents become decentralized, their interaction is subject to emergence—either positive or negative. This phenomenon is both the good news and bad news for large multiagent systems.

Analogies from nature: The autonomous and interactive character of agents more closely resembles natural systems than do objects. Since nature has long been very successful, identifying analogous situations to use in agent-based systems is sensible. For example, agents can die when they lack supportive resources. In supply-chain manufacturing, when a manufacturing-cell agent cannot operate profitably, it dies of "malnutrition." Furthermore, another manufacturing cell could come by and scavenge useful bits from the newly dead cell.

Agents can exhibit properties of parasitism, symbiosis, and mimicry. They can participate in "arms races" where agents can learn and outdo other agents. Agents can participate in sexual (and asexual) reproduction that can incorporate principles from Darwinian and Lamarckian evolution. Agent societies can exhibit political and organizational properties—whether they are organized, anarchic, or democratic. In short, nature can provide a rich trove of ideas for multiagent system design.

3 Current Notation Challenges

Representing automated systems with currently available notations is known to be problematic. The excessive need for English notes in the modeling notation is one primary indication of such inadequacies. Modeling languages that communicate to a narrow set of system developers and do not communicate to others is a problem for communication among developers in general. These limitations have already triggered a revision process in UML (known as UML 2.0), which tries to remove some of these current limitations. Furthermore, FIPA has recently launched a Modeling Technical Committee which will develop an agent-based notation called AUML (Agent-based Unified Modeling Language).

With agent-based systems, modeling languages are even more challenging because of the richness of representing agents and their systems. In this section, we discuss various aspects of agent-based systems and where graphical modeling languages might be useful to conceptualize and communicate about these systems. First, we begin by examining various aspects of intra-agent requirements. Second, we examine modeling language opportunities that represent agents interacting with other agents. Lastly, we consider the role of the environment in agent-based systems and potential areas for modeling languages.

3.1 Intra-Agent Modeling

Agents are autonomous entities and therefore must be able to manage their own thread of control. This management can consist of simple rules and procedures. More elaborate agents, however, can include belief-desire-intention (BDI) mechanisms and learning capabilities. Expressing some of these features graphically is already occurring.

Agent makeup: A common requirement for developers of agent-based systems is to specify the way in which an agent is composed. For instance, [4] suggest extensions to UML that expresses features, such as state attributes, actions, capabilities, perception, constraints, and available services.

However, agent might consist of other kinds of structures, such as classes, components, packages, as well as other agents. Here, UML class, component, and package diagrams can be employed to depict these notions.

Agent activities and goals: A new aspect that agents bring to modeling is that each agent can seek multiple goals and perform multiple tasks. These goals and tasks are pursued by the agent via the roles that the agent assumes when interacting with other agents. At first, this representation may look like no more than the equivalent to an aggregation pattern in a class diagram, which can be easily represented in UML. However, an agent's relationship with its goals and tasks is not as simple as an object aggregation. The autonomicity of an agent frequently promotes that such agents may not pursue a given goal or task, even though it might be included in its realm of specification.

Although one could extrapolate that it is easy enough to include zero as a valid quantity for a given goal/task, which would indicate that such goal/task might never be pursued, the semantics of the notation would have been changed from its original meaning.

Several existing diagrams could model some of these situations. For example, a UML activity and state diagram could depict an agent's activities flow of control or state-based nature [5]. Goals, goal hierarchies, and goal-task implications could be depicted using notations defined in MESSAGE [6]. However, these goal-related diagrams have not reached a great acceptance.

Dynamic adaptability: Different than objects, agents can have the ability to modify their own behavior. Goals and tasks can be added and removed, as new features are acquired, learned, or considered obsolete for the environment. Despite the actual methodology used to implement the learning process, the needed representation for this feature was not present on standard object-oriented modeling. Dynamic adaptability can also include when, and, where a role be acquired/learned.

Using analogy: Analogies from nature, including human social psychology can be useful to aid designing MAS. For example, modeling techniques would be useful for depicting notions such as single cell animal, the shared environment of cell structures within cell, the communication environment within a cell; a cell-to-internal-structure relation. The forthcoming section on Environmental Modeling will help with most of these concerns.

3.2 Inter-Agent Modeling

In a MAS, agents interact with other agents. Furthermore, to make multiagent systems scaleable, some form of agent grouping must be provided.

Agent interaction: Social systems consist of sets of interdependent role behaviors, providing a collective pattern in which agents play their parts, or roles. The limitations of the current notation become even more visible, when the need to represent inter-task relationships is present. To illustrate this argument, let's assume that an agent of type A can enroll as either, buyer, broker or seller in a particular negotiation, but it can only assume one of these tasks for a particular negotiation.

To further complicate the modeling, several negotiations may be active at any particular moment. Since these multiple tasks may need to access common information at the agent level, it is important to determine how access to common values is controlled and prioritized. Observe that in standard software engineering the modeler hardly ever reaches this level of detail, leaving to the implementer to guarantee correctness. In this case, however, the correctness is not at the implementer's level, but rather is an aspect of the system being modeled. UML sequence and activity diagrams [7] are one mechanism for depicting interactions using roles (See Fig. 2.). However, much still remains to be done in this area. For example, depicting role changes and role constraints still remains a challenge.

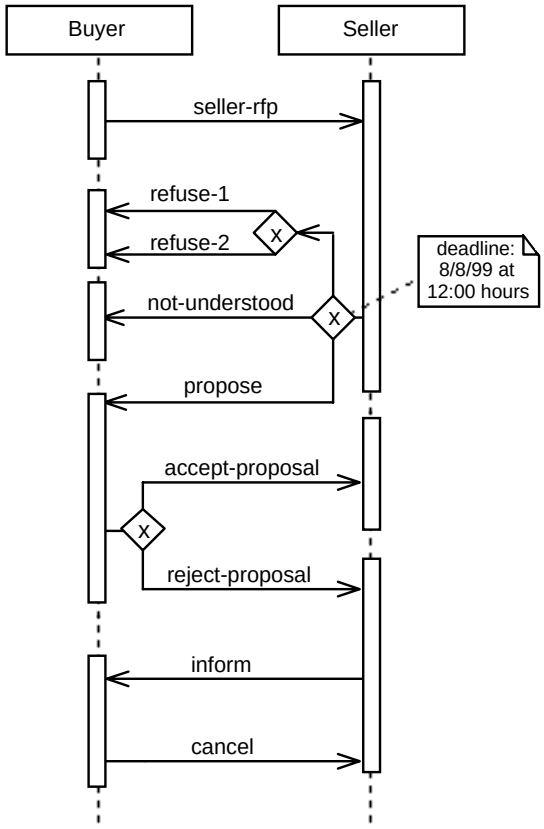


Figure 2: Interaction protocol involving buyer and seller agents.

Agent populations: Agent-based systems are no longer contained within the boundaries of single, small-agent groups. A group is a set of agents that are related via their roles, where these relationships must form a connected graph within the group. Groups can range from small "work cells" to large organizations and institutions. To meet the demands of large-scale system implementations, groups of agent must interact with other agent groups, as well as affect individual agents.

Representing groups, roles, and agent dependencies would be useful in developing MAS. Castelfranchi [8] has defined several forms of agent dependency that can be expressed graphically using a UML-based dependency diagram. Ferber [9] presents graphical approach of his AALAADIN software to represent groups, as well as their membership and interface points. However, much still remains to be done in this area. For example, a way of defining the mechanisms and environment for a group is still not very well developed. However, the forthcoming Environment Modeling section might shed some light on this.

Other: The shared environment of agents with groups, the communication environment between groups, and group-to-agent relations, is also an area for examination. It will be address in the next section on Environmental Modeling.

3.3 Environmental modeling

Another issue in which agent based systems differ from traditional OO object is in the way the agents interact with each other. Agents don't have direct access to other agents; instead they use the environment in which they are immersed to transmit messages to other agents. As an agent executes, it modifies its environment either directly (sending messages that other agents can listen) or indirectly (by altering some of the environment aspects which other agents can sense).

In this fashion the environment plays the role of a Petri dish, setting the rules with which those agents will interact. Due to its vital role, it is important to describe precisely such environment since a slight change could impact the results of the agent system in unpredictable ways. Currently there are no standardized ways to describe this important feature, and to differentiate it from the agent code itself.

Without an environment, an agent is effectively useless. Cut off from the rest of its world, the agent can neither sense nor act. An environment provides the conditions under which an entity (agent or object) can exist. It defines the properties of the world in which an agent will function. Designing effective agents requires careful consideration of both the physical and communicational aspects of their environment.

Physical Environment: The particular kind of environment that biological agents (animals and plants) require for survival is referred to as their ecological niche. While artificial agents can have different requirements for survival, they still require an ecological niche, or physical environment, to support them. The physical environment provides those principles and processes that govern and support a population of entities.

Principles: For agents, principles of the physical environment can be thought of as laws, rules, constraints, and policies that govern and support the physical existence of agents and objects. However, currently there are no modeling languages that can express the basic characteristics for an agent environment [10; 11]: accessibility, determinism, diversity, controllability, volatility, temporality, locality, and medium. Perhaps, no graphical techniques can adequately express any of these characteristics. However, some thought should go into whether or not modeling languages might be useful to the MAS developer.

Processes: In an agent environment, a primary purpose of processes is to implement the environmental principle. For example, the gravitational field is a principle that can be implemented with a process that attracts entities in a prescribed manner. In other words, the falling of an apple to earth can be regarded as the process of gravity in action. Different physical environments will be required for different kinds of agents—and vice versa. With artificial agents, much more than physics is happening because much of the environment is information intensive. In many defense-related agent systems, the information-intense environment includes satellite telemetry, body- and vehicle-based communications technology, and geographic positioning grids. In agent-based supply chains, information about orders and resources is a major component of the system.

To support the varied information requirements of such agent-based systems, a common processing platform would be useful and would consist of: application support, communication and transportation, physical linkage, agent management system, agent platform security manager, agent platform communication channel. Indeed several agent platforms have been develop to support the implementation of such agent systems (OpenCybele, JADE, Zeus, Voyager, aglets just to name a few) each with its own strengths and weaknesses.

In order to detail which features are more relevant for the MAS under development and assist implementers in selecting the correct tools, it is fundamental for the developer to be able to express the relationship of the agents with their environment as well as the structure of each agent. Again, few graphical techniques can adequately express many of these requirements. Yet, some thought should go into whether or not modeling languages might be useful to express these requirements to a MAS developer. For example, the UML deployment, component, and class diagrams might be useful here.

Communication Environment: While an agent can operate by alone, the increasing interconnections and networking require a different kind of agent—one that can communicate effectively with other agents. A communication environment provides two things. First, it provides the principles and processes that govern and support the exchange of ideas, knowledge, information, and data. Second, it provides those functions and structures that are commonly employed to enhance communication, such as roles, groups, and the interaction protocols between roles and groups. In short: The communication environment provides those principles, processes, and structures that enable an infrastructure for agents to convey information.

In rich multiagent societies (MAS), several principles are required to facilitate the communication environment. These would include: communication language, interaction protocols, coordination strategies, social policies, and culture.

An agent's communication environment provides processes that enable agents to interact productively. In particular, it must provide: interaction management, language processing and policing, coordination strategy services, Directory service, mediation services, policy enforcement service, social differentiation, and social order[1].

Providing techniques for modeling both communication principles and processes are highly important to the functioning success of any large-scale MAS. As mentioned earlier, UML sequence and activity

[1] The agent communication *channels* are defined as part of the physical environment. The communication *environment* uses those channels to convey information.

diagrams are two mechanisms for depicting interactions using roles.

4 Notation Proposition

4.1 Intra-agent modeling

In this paper, we propose the modeling of agents as classes, with a new set of associations towards their roles, which in turn can be defined as classes or components. Figure 3 shows a possible diagram to represent the relationship between an agent and its roles. In this diagram, the agent uses the UML implements association on a different manner then the original way intended by OO. Our proposed agent-modeling notion of classes has no parallel with actual implementation but rather the concept of independent structure. Hence the notion of an implementation association is somewhat different in which it qualifies the agent as capable of assuming the target role.

The diagram below has other notation propositions, which can be observed as the relationships between the roles themselves. One may observe two proposed standard associations between roles. The «prevents» association means that while an agent is performing a given role, within a context (i.e., a specific interaction between agents), it becomes illegal for such an agent to perform the other role in the same context. These associations are unilateral, which forces us to indicate twice when the association is mutual exclusive.

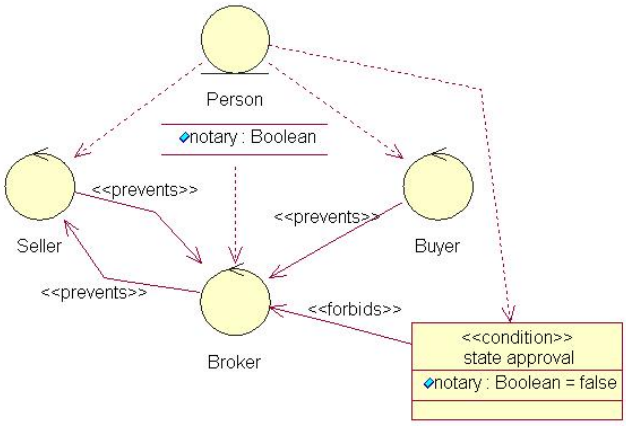


Figure 3: Proposed Class Diagram for Agents

The diagram above also demonstrates two new concepts that are important for multi-agent descriptions. The first concept is the presence of a variable. This variable does not represent a real variable in the implementation sense but rather an agent feature that is observable by its roles. The second concept is a concept of condition. A condition is a clause that holds relationships between an agent and one of its possible roles. In the example above the condition will hold true, when the agent's notary feature is false. The consequence of the condition becoming true is the associations with the roles, which in the case shown forbids the agent to assume the broker role.

There is a slight but significant difference between the «prevents» and the «forbids» association. The «forbids» association impedes the execution of a role in any context, which has a much broader effect then the former one. The dual for the «prevents» and «forbids» associations would be the «permits» and «allows» associations respectively. One can certainly anticipate the needs of other standard associations such as: obtain, reset, removes, and others, which are yet to be explored.

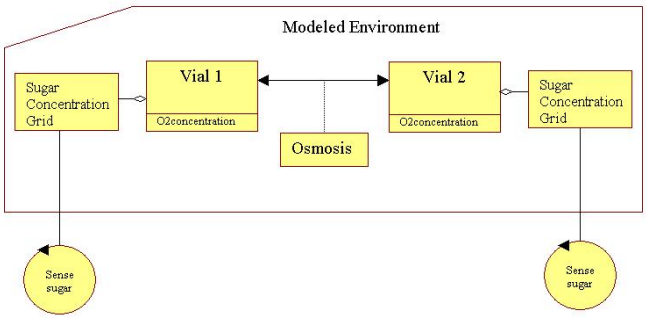


Figure 4: Class Diagram with Environment description

4.2 Environment Modeling

Our proposal for environment modeling is also based on the UML class diagram. Once more the modeling makes no inference on the implementation implication of classes but rather the encapsulation concept that they assume. In our proposed modeling the global environment is represented as wrapper around local environments. Figure 4 demonstrates a simplistic environment to simulate bacteria growth. In this environment, two sugary solutions are placed in vials that share an osmotic membrane. The relationship that describes the osmosis process between the two sub-environments is clearly defined as dependant on the mechanics of the osmosis class. Each sub-environment has its own grid that controls the amount of sugar available in a certain coordinate.

The model environment indicates that an agent has to perform a "sense sugar" role in order to receive information about the current concentration of sugar in its location. In contrast any agent in this environment immediately knows the concentration of O2 without the need to an interaction. From the aggregate symbol in the diagram above one can conclude that the grid is actually a part of the vial sub-environment, but it has encapsulated some unique behavior, as it is in this case the way the sugar diffuses in the syrup.

5 Example

The purpose of this chapter is to demonstrate how even a simple example real example can become a challenge for notion languages when the richness of the system is to be fully described such as needed when describing agent systems.

5.1 Case Study Description

The case study demonstrated is based in the United Nations Security Council resolution process and was used as a debate example in the FIPA Modeling Technical Committee.

Description: The UN Security Council (UN-SC) consists of 15 members, where 5 are permanent members and the others are rotated from the members of the United Nations according with the rules of the organization. Members become the Chair of the Security Council in turn monthly.

To pass a UN-SC resolution, the following procedure would be followed:

(1) At least one member of UN-SC submits a proposal to the current Chair;

(2) The Chair distributes the proposal to all members of UN-SC and set a date for a vote on the proposal.

(3) At a given date that the Chair set, a vote from the members is made;

(4) Each member of the Security Council can vote either FOR or AGAINST or ABSTAIN;

(5) The proposal becomes a UN-SC resolution, if at least nine members voted FOR, and no permanent member voted AGAINST (veto power).

(6) The members vote one at a time.

(7) The Chair calls the order to vote, and it is always the last one to vote.

(8) The vote is open (in other words, when one votes, all the other members know the vote)

(9) The proposing member(s) can withdraw the proposal before the vote starts and in that case no vote on the proposal will take place.

(10) All representatives vote on the same day, one after another, so the chair cannot change within the vote call; but it is possible for the chair to change between a proposal is submitted until it goes into vote, in this case the earlier chair has to forward the proposal to the new one.

(11) A vote is always finished in one day and no chair change happens on that day. The chair sets the date of the vote.

(12) There is no change in the composition of the Security Council during the entire voting process. Proposals that cannot be voted in time are automatically withdrawn and should be resubmitted (or not) when the new composition of the Security Council is reestablished.

One must observe that the procedure above was defined for a case study of agent-oriented modeling, and it does NOT necessary represents the reality.

5.2 Notation Challenges

Even in this simple system, one can identify several notions that can be problematic in modeling language representations.

The first notation challenge is to clearly represent the group organization within the Security Council amongst the several agents, (i.e., permanent/temporary members, chair) and how agents (members) join or leave their groups.

The second problem is how to demonstrate the cyclical nature of the voting process without creating a lifeline for each member and even more how to describe the temporary attributions of a member while it is occupying the “chair” role.

Other notation challenges are due to the possible combinations of allowed/disallowed membership/chair change during different moments in the process. The multitude of combinations forces us to create a modeling format that supports this flexibility and yet clearly defines which paths of execution are possible.

5.3 Proposed Diagrams

The diagrams presented in this section were our proposed solution to this study case as presented in the FIPA modeling Technical Committee forum.

Our solution for the case study presented was composed of four diagrams. The first diagram [Figure 5] presents the Security Council (SC) environment with its two groups and indicates each member by name (members were current when the solution was crafted).

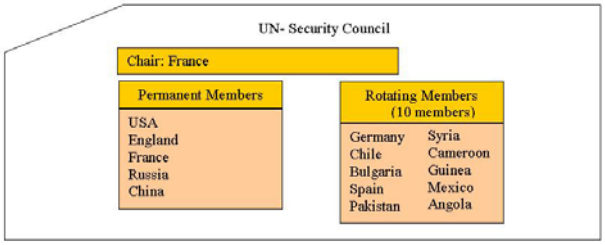


Figure 5: UN-Security Council Environment

One of the drawbacks pointed out in our solution was the lack of a process description by which temporary members are rotated (or even that this rotation is a necessary feature of the system). In order to introduce this notion, the SC environment has to be defined as a sub-environment of the whole United Nations environment. Other solutions presented in the forum, which have modeled the environment with a group membership focus, were able to express this process in a clearer fashion.

The intra-agent representation of our solution was entirely based on the functional perspective of the member agent. For a full description of the agent’s internal structure other perspectives are necessary such as goal orientation (how the agent would use the available roles to pursue a given goal), social relationship (how the instantiation of role varies the membership in the defined groups of the system) and even in case of software systems, the implementation perspective which describes each of the classes used to implement the agent and the relationship between these classes on a software engineering view.

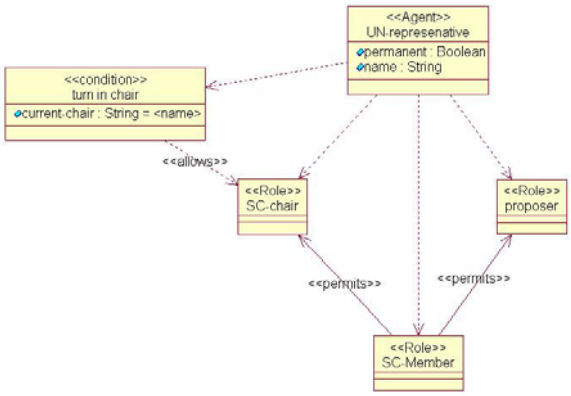


Figure 6: Intra-Agent functional description

In object-oriented systems, typically only the implementation perspective is used and notions of the functional perspective are merged into the diagram. Due to the complexity of agent systems (and its use to explain and predict model behaviors in non-software oriented domains) a clear separation and indication of the perspective of the diagram becomes quintessential. To our knowledge this kind of diagram (with small nomenclature and notation changes) seems to be the most homogenous between the ones used to describe agents systems.

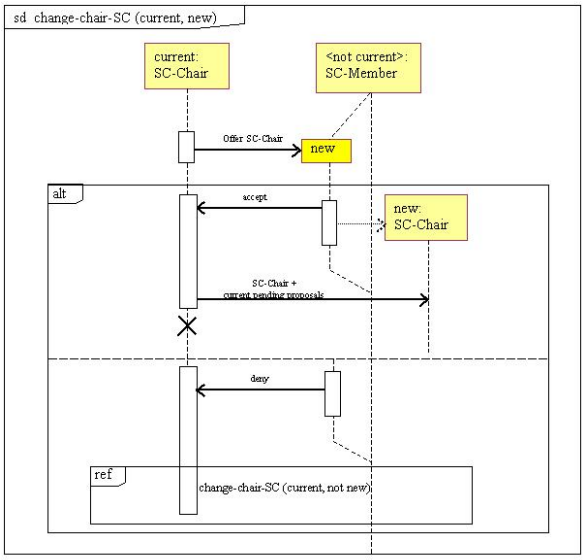


Figure 7: Chair rotation interaction diagram

Figure 7 and Figure 8 show the chair rotation process and the proposal voting process in an interaction diagram format (sequence diagram in UML). In our proposal we have tried to keep the notation as close as possible with the newer version of UML (2.0), altering and extending only when necessary.

One of the extensions was the usage of parameters to define a specific individual in a lifeline that represents a group in which the individual is member. The usage of agent conditions (current chair) or message-defined values allows the representation of the group as a whole in the lifeline, and at the same time isolates the addressed individual in the group, promoting a temporary bifurcation of the lifeline.

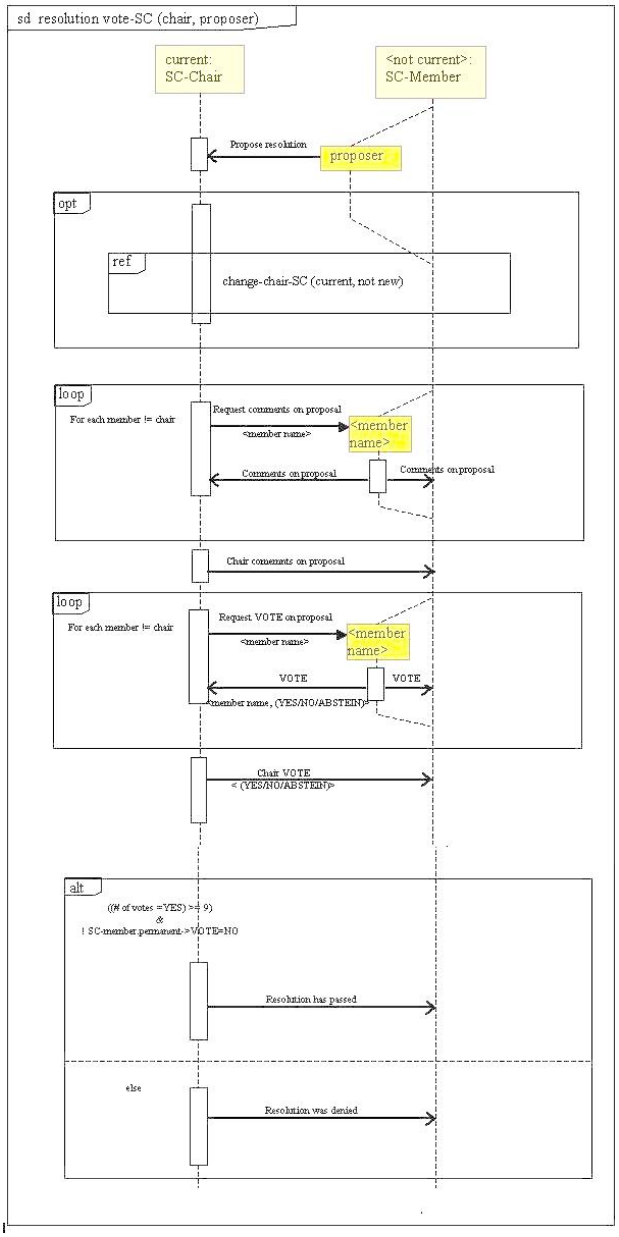


Figure 8: interaction diagram for proposal voting

The lifeline bifurcation (present in UML 2.0 without parameters) has been criticized as being visually cumbersome when several blocks (alt, loops, …) are involved.

The second extension is expressed in Figure 7, to indicate the change/add of role in which a SC-member becomes the new chair of the Security Council.

The final extension is only to create the optional block representation (marked by an opt label in the block construction). This type of block, which does not exist in UML 2.0, indicates that actions within the block may or not happen (as a block). This simple extension allows the consolidation of two very similar interaction paths and hence the simplification of the overall interaction diagram.

Discussions with the FIPA modeling technical committee have raised the concern that the relationships between different interaction diagrams are not clear in our solution. Other authors in the forum have presented Workflow/Activity based diagrams that were developed to present the overall scheme between these diagrams.

6 Conclusion

In this paper we have presented some of the challenges of modeling and notation of agent based systems and how they differ from standard object oriented systems. We have also proposed a notation format for the presented challenges that are compliant with an extended view of UML.

This paper has no intention to try to determine the best notation for agent systems. The intention is rather to present the need and stir the debate on this issue that is currently active in the Agentlink and FIPA forums.

Acknowledgement

We would like to acknowledge NASA for funding this effort in standardization of AUML notation as part of the project under contract NAS2-02003 and the collaboration of NASA's technical representative Ms. Michelle Eshow in our efforts.

Our thanks to Radovan Cervenka, Hong Zhu and Misty Nodine, for their collaboration in the definition and discussion of the study case presented which was extracted from the discussion in the FIPA Modeling TC forum, where each researcher presented their own solution.

References

[1] HPLabs, http://www.hpl.hp.com/agents/

[2] BritishTelecom- http://more.btexact.com/projects/agents.htm

[3] Parunak, H. Van Dyke, "'Go to the Ant': Engineering Principles from Natural Agent Systems," Annals of Operations Research, 75, 1997, pp. 69-101.

[4] Huget, Marc-Philippe, "Agent UML Class Diagrams Revisited," proceedings of the AgeS 2002 Workshop, Bolognia, 2002.

[5] Odell, J., H.V.D. Parunak, and B. Bauer, Representing Agent Interaction Protocols in UML, in Agent-Oriented Software Engineering, P. Ciancarini and M. Wooldridge, Editors. 2001, Springer: Berlin. p. 121-140.

[6] Evans, R., et al., MESSAGE: Methodology for Agent-Oriented Software Engineering. 2001, EURESCOM Project P907, Deliverable 3.

[7] Odell, J., H.V.D. Paranak, and B. Bauer, "Extending UML for Agents," in Proc. of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, G.W. Yves Lesperance, and Eric Yu, Editor. 2000, workshop proceedings: Austin, TX. p. 3-17.

[8] Castelfranchi, C., "Engineering Social Order," Nordic Journal of Philosophical Logic, 2002. (to appear).

[9] Ferber, J. and O. Gutknecht, "A meta-model for the analysis and design of organizations in multi-agent systems," in Third International Conference on Multi-Agent Systems (ICMAS'98). 1998. Paris, IEEE Computer Society.

[10] Weiss, G., ed. Multiagent Systems: A Modern Approach to Distributed artificial Intelligence. 1999, MIT Press: Cambridge, MA.

[11] Russell, S. and P. Norvig, Artificial Intelligence: A Modern Approach. 1995, NJ: Prentice-Hall

An XML-based Serialization of Information Exchanged by Software Agents

Sînică Alboaie
Institute of Theoretical Computer Science, Romanian Academy and Iaşi branch
abss@iit.iit.tuiasi.ro

Sabin Buraga and Lenuța Alboaie
Faculty of Computer Science, "A.I.Cuza" University of Iaşi, Romania
{busaco,adria}@infoiasi.ro – http://www.infoiasi.ro/~busaco/

Keywords: Software Agent, Serialization, XML, Distributed Resources

Received: July 15, 2003

In this paper, we present an agent-based object-oriented solution to access the Web distributed resources. We describe Omega *– an agent framework viewed as a hierarchical space of a set of distributed objects that models the Web resources. Also, we propose an XML-based model that can be used as a universal manner for serialization of the objects processed by the (mobile) agents. The serialization mechanism can use the Simple Object Access Protocol (SOAP) serialization facilities, also.*

1 Introduction

The primary goal of Tim Berners-Lee's vision of the Semantic Web [5, 12] is to develop different mechanisms to automatically exchange, by the software entities, knowledge on the Web instead of the conventional manner used for accessing distributed resources.

To do this, computer scientists need to achieve the following:

- To understand the semantic mechanism of all kinds of queries, and what kind of components the process of questioning the Web formally consists of;
- To rigorously capture, represent or symbolize the knowledge contained on the Web.

To accomplish this goal, we are designing and implementing a framework – *Omega* [2, 3] – for agent software development viewed as a tree-like space of a set of distributed objects that models the Web resources by using XML (Extensible Markup Language) [7, 9] constructs. The *Omega* system offers a flexible framework for building agent-oriented distributed applications on the Web (see details in section 2 of this paper).

To assure the Web scalability, independently designed programs (especially Web agents) must be able to exchange and to process the meaning of data and metadata in an independent manner. Semantic interoperability can be completed only if different users (agents, tools, other Web clients, etc.) interpret XML – the actual *lingua franca* of the World-Wide Web computing entities – documents in the same way.

The *Omega* framework offers an addressing space for the Web objects and a mechanism for remotely accessing the Web distributed resources (objects). In section 2.2 of the paper we'll present the internal architecture of the *Omega* system, its functionality and base classes. A script-like language is provided, in order to implement an active (execution) part of the system and to integrate the *Omega* object space with notions such as execution thread, function, instruction, data types (see details in section 2.3).

To enable the flexible querying and accessing mechanisms about the distributed Web resources, we must offer a facility for serialization – in an independent way – of the data and metadata (objects) processed by the *Omega* agent system. In section 3, we investigate different possibilities of serialization given by the XML family of mark-up languages [9, 24]. Some of the drawbacks due of the lack of a description language regarding the objects' properties can be elegantly resolved by XML. Also, a SOAP-based serialization mechanism is presented and some advantages of the SOAP protocol are discussed (see section 3.2).

Even our approach can be used in the context of Web services discovery and description infrastructures, the paper does not intend to discuss these issues.

From the authors' point of view, the serialization of the Web objects can be considered as a flexible way to exchange information between software agents. Related multi-agent environments are presented in section 4 and some possible further development directions are exposed in the last section of the paper.

2 *Omega* Agent Framework

2.1 Motivation

We can consider as the fundamental resources that computers expose to the software components (i.e. operating system, applications) or users the following items: computing capabilities, (volatile or non-volatile) memory, local and remote data (documents), metadata (different descriptions about several properties of the resources: content, structure, layout/interface, dynamics, security issues, etc.).

Of course, there are other modalities to describe these properties without using XML-based assertions, but with the penalty of the platform and software independence. Obviously, these documents (including XML resources) are made to be read and processed in a distributed system (the Web itself). To easily access and obtain the knowledge contained by a specific document, a universal mechanism/model – based on the XML family – must exist to accomplish that. This is the seminal idea of the Semantic Web [5, 12].

WWW Space as a Distributed Hypermedia System

Also, the World-Wide Web space can be viewed as a distributed hypermedia system that uses Internet technologies (i.e. TCP/IP protocol family) – a global system of heterogeneous networked computers. Advances in networking and Internet/Web technologies are leading to a network-centric computing model, and the Web and Internet itself, of course, are evolving into the infrastructure for global network computing. By populating this infrastructure with object-based components and combining them in various ways, we can enable the development and deployment of interoperable distributed object systems on the Web.

The object model provides the ability to mimic real world processes in a fluid, dynamic and natural manner. The Web space allows for objects to be distributed to servers thereby centralizing access, processing, and maintenance, provides a multiplexing interface to distributed objects, and allows thin-clients (e.g., mobile phones or handheld devices). We can safely now state that **Web + Object** integration is a viable reality [24]. This is emphasized by different software organizations and companies – especially in the e-business domain – that are using Web-enabled distributed object technology, in the form of intranets and extranets, to solve their computing problems, and the emergence of an industry that provides Web and object interfaces to distributed object tools [4].

From the CGI Approach to a Distributed Object Infrastructure

But the Web didn't start out this way. Network-centric object computing is the result of a logical technological evolution. As originally conceived, it was driven by hypertext documents called Web pages or HTML documents [5, 9]. Initially, Web pages had static content (rich text and graphics at first, complex multimedia information later), and were interlinked. Browser applications running on user PCs or workstations were used to retrieve documents stored on Web servers. Helper applications supplemented the browser, handling other document types such as Word, PostScript, PDF (Portable Document Format) or different graphics, video, and audio formats. Web pages soon begun to include dynamic content as helper applications, called *plug-ins*, were integrated into the browser and CGI (Common Gateway Interface) scripts enabled users to input data to a Web server and access Internet services (i.e. data queries). Finally, programmatic content was added, on the client side, via Java applets, VBScript and JavaScript programs, to provide further interactive functionality and modify content in-place. These languages and techniques enable richer documents (e.g., animation and Web forms generated on-the-fly). Note that programmatic content can also include server-side execution of code such as accessing a remote database service (i.e. SQL queries) via specific Web application platforms (from CGI programs to PHP, ASP.NET or ColdFusion applications).

Prior to the addition of programmatic content, the Web was based on a client/server computing model which lacked scalability, common services, security, and a development environment needed to develop and deploy large-scale distributed applications. CGI scripts are not scalable because each requires a separate server-side process to handle each client request, services are limited to accessing database servers via CGI scripts, transaction information (such as credit card information) is not encrypted, and the programming model offered by HTML/HTTP using CGI and a three-tiered system is limiting.

With the advent of Java, and the distributed object infrastructures CORBA/IIOP and OLE/DCOM, the stage was set to evolve the Web from a document management system to a platform for distributed object computing and electronic commerce.

Bringing distributed objects to the Web offers the following advantages (to name only few of them):

- extensibility (e.g., for applications, services, and APIs built from objects, objects can easily be replaced or added);
- cross-platform interoperability;
- independent software development;
- reusable software components;
- componentware;
- network services;
- better utilization of system resources.

Existing legacy applications can even co-exist with distributed objects through the use of object wrappers. The interface could either be the client browser or browser-like with super-positioned distributed object infrastructures.

Mobile Agents

An important step towards *Internet/Web Computing* is represented by the mobile computations. A mobile

object, usually called an *agent* when operating on behalf of a user, is a downloadable, executable object that can independently move (code and state) at its will – the mobile agent is not bound to the system in which it began the code execution and can travel from one node (host) on a network to another. Agent technology can be considered as a natural extension of object technology; conceptually, agents support a much richer and complex range of capability than objects, such as adaptability, cooperation, autonomy, negotiation and delegation [6, 17]. These capabilities give the possibility to build a sophisticated, expandable, maintainable, and distributed computing environment.

Mobile agents present the following main attributes [6, 15]:

- *reactivity* – the ability to respond to changes within agent environment;
- *autonomy* – the mobile agent is able to exercise control over its own actions (decisions);
- *goal-oriented* – the agents have a planned itinerary, they do not simply act in response to the environment;
- *communicative* – the ability to communicate with other agents, by exchanging information (knowledge); in this sense, agents present a collaborative behavior is order to achieve a common goal with other agents of the environment;
- *temporal continuity* – persistence of identity and state over long periods of time;
- *adaptability* – being able to learn and improve with experience;
- *mobility* – the mobile agents can transport themselves from one machine to another, in a self-directed manner.

Mobile agents provide a way to think about solving software problems in a networked environment that fits more naturally with the real world. Mobile agents can be used to access and manage information that is distributed over large areas [6].

The main benefit is that the software components can be integrated into a coherent and consistent software system – e.g. a multi-agent system – in which they work together to better meet the needs of the entire application (utilizing autonomy, responsiveness, pro-activeness and social ability).

The mobile agent architecture provides the "framework within which mobile agents can move across distributed environments, integrate with local resources and other mobile agents, and communicate the results of their activities back to the user. This framework can then be used to build mobile agents that perform user-driven tasks to fulfill distributed information management goals." [6]

Taking this notion further, the mobile agents could be used to monitor the network activities and provide input to QoS (Quality of Service) and global optimization mechanisms. They could be used during negotiation (with representative agents) to solve different constraint optimization problems.

One key research area is to provide security against malicious agents (who intend to access local resources or can carry a virus) and malicious hosts (who can alter the agent code/state or read private information).

Current mobile agent systems [17] – available as commercial or open-source applications – are implemented in different programming languages, such as C++, Java, Tcl, Scheme or Python.

2.2 Internal Architecture of the *Omega* System

Overview

Omega is an agent-based system that offers a tree-like addressing space for the Web objects and different techniques to remotely access the Web distributed resources (viewed as objects) [2, 3]. Each object processed by *Omega* can be viewed as a collection of objects included in that one. The links (edges) between the vertices of the tree are given by the aggregation relationship exposed by the object-oriented methodologies.

To emphasize the aggregation relationship, we attach to each object a name or an index, and in this way we can uniquely refer each object of the tree by its name/index (viewed as an identifier). Each object will have a unique list of the identifiers that represent its "address" in the addressing space used by the *Omega* agents. An identifier can be considered as an `IName` object (at the implementation level, an `IName` object can be viewed as an object-tree path or a list of object identifiers). By using a tree of objects, we can structure more easily the distributed resources for a given local web (such as a cluster or an intranet).

Functionality

We choose to use an interpreted environment for our multi-agent model and distributed object structure. Using such an environment, it was easier to consider serialization and various execution control mechanisms [11] which are contributed to the implementation of the *Omega* distributed object system.

Omega offers a distributed object structure, and its initial goal was to determine some good representations of data, types, instructions, functions and objects of an object-oriented language that can be used as a programming language for mobile agents. The result of this effort is a system written in C++ that is able to unify the notions behind the object-actor duality, namely the duality between passive and active objects [1]. From this point of view, *Omega* offers an infrastructure able to support Web-based distributed applications [18] (e.g., software agents used in clusters or Grid).

As an example, let us consider the problem of a system in which someone from a location *A* wish to obtain in real time data from another location *B*. There is more than one solution, and we present here just two possibilities [2]:

- Using the multi-agent paradigm, we create two agents in *A*: an agent for the information point, and an agent to be sent at the location *B* in order to obtain the information needed to be communicated to the location *A*. This approach is used in the design of *Omega* [3].
- Another way of solving this problem is to create a Web site at *B* (providing different server-side solutions [9] – i.e. CGI scripts or Java servlets) or a client/server application using a proprietary (TCP/IP-based) protocol [10].

We can observe that the first solution (the multi-agent approach) is more scalable and closer to follow certain good rules for programming design.

By using the multi-agent paradigm, a system can be easily divided into small entities with control over their interactions. Moreover, we can get a more flexible and adaptable approach (in our example, we can have a more adaptable way of presenting the information at the location *A*). This flexibility is a part of the client task, as opposed to the Web approach where we require more tasks for the server. In the case of using a client/server solution (with a proprietary communication protocol), some problems come from the high cost of the system design and maintenance.

The solutions that use C++ (networking, DCOM, CORBA) or even Java/C# are quite complicated and they are, in many cases, inappropriate for an open system, as the Internet – and Web, also – is. At the moment, for the generic problem of our example, a client/server solution is more popular in industry (in many cases based on HTML or XML). The later approach is adopted by Web services [18, 25] scenarios, also.

From the object-oriented paradigm's perspective, *Omega* can be seen as an object hierarchy that ensures a unitary way of programming, with an implementation of a name-service (presented in [2]) that is consistent for the resources (objects) that it makes available. The *Omega* system offers serialization mechanisms and garbage collection, also.

***Omega* Classes**

The `IObject` class is the base-class for every other class that has memory regions stored within a local system. Every object and function that needs a store space in *Omega* will use `IObject`. In this way, *Omega* assures a space model provided by a common distributed memory. This model is based on the existence of a given node of an `IObject`'s tree, which is easily addressable from the network.

Omega system offers a number of object types which provide functionality to the following classes:

- *String* class,
- *Number* class,
- *List* class,
- *Control* agent-execution class (i.e. support for virtual threads, scripting languages etc.).

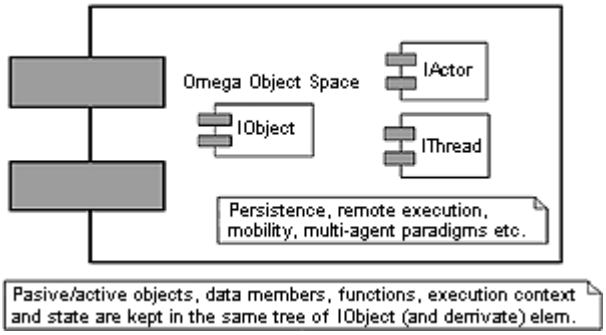


Figure 1: *Omega* objects

Within the *Omega* framework, data types are represented by different classes such as `IString`, `INumber`, `IOmegaStack`, `IOmegaList`, `IOmegaQueue` that are derived from the `IObject` generic class (see also Figure 1).

Omega offers two categories of data types [3]:

- *Simple data types* – have no components (i.e. `INumber`, `IString`, etc.)
- *Compound data types* – represent a mix-up of two or more simple types (e.g., `IName`, `IOmegaList`, `IAThread`).

A compound data type can be considered as an "array" or a "struct" (very similar with the `struct` used in the standard C language).

In our approach, the string data type (`IString`) is not similar to the common concept of the "string"' type (present in all modern programming languages). At the implementation level, *Omega* system will use for `IString` another manner to store the content of a string (we do not use XML Schema's `xsd:string` – see details in [13]).

2.3 Omega Language

For the object system presented above, we provide an active (execution) part, which is the implementation of a scripting language that is using *Omega* objects. We can integrate the object space with notions such as execution thread, function, instruction, data types to be modeled with the help of `IObject` abstraction. The execution threads represented by an `IAThread` object (actor thread) will have a current execution context in which it can keep the local names and a global name list of the task (a task has more execution threads, some objects have attached execution threads, and they have the same name list from the task they belong to).

To simplify the development of a high-level control language, we are started from a data-type model that had `IString`, `INumber`, `IThread`, and `IObject` as base types and various types derived from `IOmegaActor` (this class is derived from `IActor`). The system is able to initialize and execute `IOmegaActor` objects.

Therefore the *Omega* object environment and the *OmegaKernel* mini-interpreter provide [2]:

- A *data model* (base type-system, the construction of new objects),
- An *address space* (every object has its own address consistent at the Internet – by using the TCP/IP stack – level),
- Techniques to implement the *high-level programming level statements* (e.g., `if`, `while`, or `goto`).

The *Omega* system is able to execute small ("scripting") programs. We present below such a program called *test program* – new `IObject`s are created. At runtime everything is reduced to a creation of new `IObject`s in the distributed space of objects.

```
# A simulation of while statement
OmegaTrace ("Test begin")
# OmegaTrace could be used
# for debugging purposes
BeginActor (SimulateDoWhile)
# BeginActor initializes
# an independent actor thread
NewINumber i 0
label begin
Inc i
OmegaTrace ("i++ in SimulateDoWhile")
LessThenGoTo i 2 begin
EndActor
SimulateDoWhile ()
OmegaTrace ("Test end")
```

The language provided by the *Omega* framework is similar to an assembler language and may be easily extended with other instructions. The main syntactic construct is similar to a function (method) call. An important step was to create a mechanism for representing data structures, statements and objects under the same abstraction (`IObject`) that is a network shared entity.

3 Serialization Mechanism

All classes derived from `IObject` must implement the *serialization* (*marshalling*) and *deserialization* (*unmarshalling*) methods. The process of building of the new data types is based on the fact that an `IObject` has a member of the `IOmegaList` type. That member contains associated links which are instances of the derived classes. In this manner, the serialization of the new types of objects can be automatically accomplished by *Omega* via members' serialization and the call of the overloaded own methods. Of course, for several types of objects – e.g. `IOmegaSocket` used for usual BSD-like socket operations [10], such as `bind()`, `listen()`, `accept()` or `connect()` – the serialization and deserialization activities can not be viewed as a proper solution.

For each access to a sharable object, a *proxy-object* is created, using the RPC (Remote Procedure Call) mechanism [9]. This proxy-object is placed in the same tree of the target object. In the tree of the accessed object, a *stub-object* is created, too. The stub will contain meta-descriptions about the sharable object and will be derived from `IObject`. The stub-object will be a member of the sharable object, to allow us to remotely access the stub. In this way, the system will be able to keep updated versions of the different object trees. To obtain the serialized form of an object, the RPC-like mechanism is able to transmit the URI (Uniform Resource Identifier) [9, 25] of that object. As a response, the system will get the serialized forms of the object and of the proxy-object as well, if it is possible. The *Omega* system is responsible to regularly update the proxy-objects.

The object serialization does not imply the serialization of the whole sub-tree that has as root the object in cause. For an object, only the serialization of the object itself and of the `IName` list of its children is done.

3.1 XML-based Serialization

The process of the *Omega*'s object serialization uses XML-based constructs. We use the XML namespaces defined by the XML Schema specification (see [13]) to retain the primary types of the data exchanged by agents in the serialization and deserialization processes.

An example is following (we are using an `IString` object):

```
<?xml version="1.0"?>
<IString>
   <name xsi:type="xsd:string">
      Hello from Omega
   </name>
</IString>
```

The *Omega* encoding style is based on the usual XML Schema's data types [13]. All data types used within the *Omega* system of agents must either be taken directly from the XML Schema or derived from *Omega* data types (see section 2.2).

The XML Schema specification (see *Datatypes* section from [13]) does not offer the possibility to express data types as XML elements, but only as attributes. To address this, the *Omega* framework declares a schema, called `OMEGA-ENC`, used to define an XML element for each data type (see the example below).

```
<OMEGA-ENC:int id="int1">
   33
</OMEGA-ENC:int>
```

Example
An example of *Omega* object serialization follows:

```
<element
   name="local_address_type"
   type="...">
  <simpleType
     name="local_address_type"
     base="xsd:string">
     <enumeration
         value="tree_id" />
     <enumeration
         value="unique_name" />
  </simpleType>
</element>
<element
   name="local_address"
   type="..." />
  <complexType
       name="local_address">
    <element
       name="la_type"
       type="local_address_type" />
    <element
       name="la_value"
       type="xsd:string" />
  </complexType>
</element>

<IName>
  <IOmegaDomain>
      ...
  </IOmegaDomain>
  <!-- info about local addr. -->
  <local_address>
     <la_type>
         tree_id
     </la_type>
     <la_value>
         1
     </la_value>
  </local_address>
  <local_address>
     <la_type>
         unique_name
     </la_type>
     <la_value>
         member_name
     </la_value>
  </local_address>
  <!-- other similar constructs... -->
</IName>
```

These XML elements could be used to extend the functionality of the *Omega* system with new data types.

We can note the *Omega* system only proposes the presented XML-based manner of object serialization, but does not interdict other mechanisms – e.g. SOAP-based serialization – to be adopted for data serialization.

3.2 SOAP-based Serialization

SOAP – or other protocols that use the RPC over XML approach (e.g., XML-RPC) – will be used to transport the serialized data. SOAP looks to be the right solution because of the great support it gets from different companies and organizations.

Short Description
SOAP (*Simple Object Access Protocol*) [14, 25] is a simple lightweight protocol used for XML-based structured and strong-type information exchange in a decentralized, distributed environment. The protocol is based on XML and consists of three parts:

- An envelope that describes the contents of the message and how to use it;
- A set of rules for serializing data exchanged between applications;
- A procedure to represent remote procedure calls, that is the way in which queries and the resulting responses to the procedure are represented.

Similar to object distribution models (e.g., IIOP and DCOM) [4], SOAP can invoke methods, services, components, and objects on remote servers. However, unlike these protocols, which use binary formats for the calls, SOAP uses a text format (Unicode), with the help of XML, to structure the nature of the exchanges.

SOAP can generally function with several protocols, such as FTP (File Transfer Protocol) or SMTP (Simple Mail Transfer Protocol), but it is particularly well-suited for the HTTP (HyperText Transfer Protocol) [9, 25]. It defines a reduced set of parameters that are specified in the HTTP header, making it easier to pass through proxies and firewalls. The use of SOAP over HTTP also enables resources already present on the Web to be unified by using the natural request/response model of HTTP protocol. The only constraint is that a SOAP message via HTTP must use the MIME (Multi-purpose Internet Mail Extensions) [9, 25] type `text/xml`.

Also, SOAP protocol can help in activities of message exchange and routing and agent communication by integrating well-known actual standards (e.g., The Foundation of Intelligent Physical Agents – FIPA agent standard [23]).

The actual SOAP implementations are available for a broad range of programming languages, such as C++, C#, Java, Perl, PHP or Python.

SOAP vs. CORBA
Although SOAP was initially intended as a remote method invocation protocol running over the Internet and using XML messaging, the SOAP protocol is not just another Common Object Request Broker Architecture (CORBA) [4, 20].

SOAP presents the subsequent significant improvements [14, 16]:

- *Human readability* – SOAP does not expose a binary format like CORBA Internet Inter ORB Protocol (IIOP); even if SOAP is mainly projected to be read by machines and to give support for Web services, human readability is very useful for debugging purposes and rapid and simple implementations;
- *Simple installation* – because SOAP is based on HTTP and XML, the protocol can be implemented with slight effort by using existing processing libraries for XML and HTTP; contrary, CORBA requires complex software packages and does not provide a commonly accepted bootstrapping mechanism.

The SOAP protocol has the potential to become the connecting point between heterogeneous distributed platforms and architectures, such as Sun ONE, Microsoft .NET, Perl or PHP scripting applications.

SOAP Data Model

SOAP is based on a simple object-oriented data model. The SOAP data model consists of structured objects having certain properties and a type. The SOAP specification allows, through a set of unambiguous rules, alternative syntax forms for embedded and referenced objects. Objects can be embedded if there exists only one referenced to them; otherwise they are linked [14, 25].

SOAP does not provide its own schema language. For this, the protocol uses XML Schema [13] for validation of the syntactical correctness of SOAP serialization model. Also, SOAP serialization fits fine into Unified Modeling Language (UML) modeling [20]. Even if SOAP describes instance serialization only, the UML meta-model can be utilized to serialize UML models using SOAP serialization syntax [16]. This can be a helpful feature in the activity of multi-agent system design.

In [16], the SOAP-based serialization mechanism is discussed in conjunction to Resource Description Framework (RDF) and the related Semantic Web activity. Also, RDF assertions can be used to store certain metadata about existing objects [8].

Example

A short example is following, when a request to invoke a remote method of an object is made and a response that contains the result is returned. The invoked method returns the services provided by a given node (agent) of the system.

The SOAP request can be (first five lines are HTTP header fields followed by the SOAP envelope marked-up in XML; the `SOAPAction` field specifies the action to be executed on the remote site):

```
POST /omega/interface HTTP/1.1
Host: 193.231.30.197
Content-type: text/xml
Content-length: nnn
SOAPAction: urn:omega.ro:Omega:#getSrv
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=
     "http://schemas.xmlsoap.org/
              soap/envelope/"
      SOAP-ENV:encodingStyle=
        "http://schemas.xmlsoap.org/
            soap/encoding/">
   <SOAP-ENV:Body>
      <o:getSrv
         xmlns:o="urn:omega.ro:Omega">
         <node ip="193.231.30.225">
            thor.infoiasi.ro
         </node>
      </o:getSrv>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

A possible response (on success) can be the following (first three lines denote the response given by Web server, followed by SOAP data – in this case an XML-based document that contains the list of the existing agents and additional information about them):

```
200 OK
Content-type: text/xml
Content-length: mmm
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=
     "http://schemas.xmlsoap.org/
              soap/envelope/"
      SOAP-ENV:encodingStyle=
        "http://schemas.xmlsoap.org/
            soap/encoding/">
   <SOAP-ENV:Body>
      <o:listSrv
         xmlns:o="urn:omega.ro:Omega">
         <service desc="...">
            <stateInformation>
                ...
            </stateInformation>
            <securityInformation>
                ...
            </securityInformation>
            <transportProfile>
                ...
            </transportProfile>
         </service>
         <service desc="...">
            <stateInformation>
                ...
            </stateInformation>
            <securityInformation>
                ...
            </securityInformation>
            <transportProfile>
                ...
            </transportProfile>
         </service>
         <!-- other information -->
      </o:listSrv>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Using *gSOAP* for Data Serialization

We are using an existing tool named *gSOAP* [21], which is able to generate the code for serialization from a user-defined specification.

Most toolkits for C++ Web services adopt a SOAP-centric view and offer APIs for C++ that require the use of class libraries for SOAP-specific data structures. This often forces a user to adapt the application logic to these libraries. In contrast, the *gSOAP* compiler tools provide a unique SOAP/XML-to-C/C++ language binding to ease the development of SOAP/XML Web services and clients in C and/or C++ languages.

The compiler enables the integration of (legacy) C/C++ programs, embedded systems, and real-time software in SOAP applications that share computational resources and information with other SOAP applications, possibly across different platforms, language environments, and disparate organizations located behind firewalls.

4 Related Work

Although there is not a formal framework for multi-agent systems development, due to dependence on application domains, it has been that the construction of these systems requires a different approach from that of conventional software systems development.

We are aware of multiple platforms developed both in academia and software industry companies [17]. This confirms that many computer scientists are considering the agent-oriented software as a possible paradigm, designed and implemented especially in very dynamic environments (such as World-Wide Web space). We can give different examples of frameworks and tools used to develop multi-agent systems (for more details, see [17]), some of them using the Internet open standards:

- *Tryllian's ADK (Agent Development Kit)* – an agent-based business integration platform, designed and built in Java, XML and JXTA with a modular architecture and a unique mobile component approach;
- *Toshiba's Bee-gent (Bonding and Encapsulation Enhancement Agents)* – a CORBA-based communication framework intended to provide co-operative processing in the advanced network society;
- *FIPA-OS* – a Java component-based layered toolkit enabling rapid development of FIPA (Foundation for Intelligent Physical Agents) compliant agents;
- *Grasshoper* – an open-source CORBA-based platform that allows software agents to move between different fixed and wireless computing systems and to execute various tasks in the process; this platform provides support for MASIF (Mobile Agent System Interoperability Facility) – a standard specification developed by the Object Management Group (OMG) [20];
- *JADE (Java Agent DEvelopment Framework)* – a widely used agent platform that can be distributed across heterogeneous machines and that can be configured via a remote GUI (Graphical User Interface);
- *Xraptor* – a simulation environment for continuous virtual multi-agent systems written in C++ for UNIX platforms that allows studying the behavior of agents in different 2- or 3-dimensional worlds.

Another interesting approach is *Agentcities* – a world wide initiative designed to help realize the commercial and research potential of agent based applications by constructing an open distributed network of platforms hosting diverse agents and services [19].

However, the existing implementations have not convinced the whole community or do not cover or provide certain facilities desired by programmers or final users. Some proprietary solutions, though well developed, are not built as open systems and can not be easily extended or modified. On the other hand, we were not impressed by the available open-source platforms. Therefore, from the authors' point of view, it was more useful and interesting to design and implement new systems, hoping that they will cover and combine better features.

The existing multi-agent platforms use different approaches for communication between agents, by using low-level communication protocols (TCP/IP, SMTP and HTTP) or standard high-level languages – such as KQML (Knowledge Query Manipulation Language) [6, 17]. One of the noticed difficulties is to design a platform-independent inter-agent communication language.

The *Omega* system presents an advantage, by adopting an XML-based platform-independent approach in serialization and exchanging information between agents. The SOAP model is more flexible and easy to use than CORBA or DCOM solutions. Some of the *Omega*'s facilities could be also integrated, for example, into the MAIS (Mobile Agents Information System) – a platform for creating dynamic clusters [15].

5 Conclusion

We have used the design principles of the distributed systems to develop our own software platforms and ideas related to the multi-agent paradigm and actor spaces (see also [1, 11]). From this point of view, the *Omega* project represents an infrastructure able to support the agent-oriented programming and to assure an XML-based flexible way for object serialization.

The paper focused on different platform-independent methods of exchanging information between the entities of a multi-agent infrastructure – *Omega* – presented in section 2.2. The *Omega* project can be viewed as a platform for developing distributed object

middleware components [4]. *Omega* proposes a distributed object structure, and its initial goal was to determine some good representations of data, types, instructions, functions and objects of an object-oriented language that can be used as a programming language for mobile agents. The language provided by the *Omega* environment is a simple scripting language described in section 2.3.

To proper exchange information between the entities of a multi-agent system, an XML-like messaging solution is proposed. All classes within the *Omega* system must implement certain serialization (marshalling) and deserialization (unmarshalling) methods. The process of the *Omega*'s object serialization uses XML-based constructs and is detailed in section 3.1. Another method for object serialization is the use of SOAP-based serialization (see section 3.2).

Using these approaches, the *Omega* multi-agent system could integrate different Web services or could be integrated into complex distributed architectures such as Grid [18].

As a further research work, the proposed model for serialization will be used to exchange knowledge (using RDF, DAML+OIL or OWL assertions, for example) [8, 9, 12, 25] between intelligent Web agents. This research direction can be viewed as an effort to give support for Semantic Web projects [12, 21].

Also, we intend to experiment an XML-based version of the *Omega* language to be used to exchange mobile code of the software agents coded within the *Omega* framework.

References

[1] G. Agha, C. Callsen (1993) Actor Spaces: An Open Distributed Programming Paradigm, *Proceedings of the 4th ACM Symposium on Principles and Practice of Parallel Programming*, ACM Press

[2] S. Alboaie, S. Buraga, L. Alboaie (2002) An XML-based Object-Oriented Framework for Developing Software Agents, *Scientific Annals of the "A.I. Cuza" University*, Computer Science section, Tome XII, "A.I. Cuza" University Press, Iaşi, Romania, pp.109--134

[3] S. Alboaie, G. Ciobanu (2002) Designing and Developing Multi-Agent Systems, in *International Symposium on Parallel and Distributed Computing (ISPDC) Proceedings*, Scientific Annals of the "A.I. Cuza" University, Computer Science section, Tome XI, "A.I. Cuza" University Press, Iaşi, Romania, pp.142--153

[4] D. Bakken (2001) Middleware, in *Encyclopedia of Distributed Computing*, Kluwer Academic Press

[5] T. Berners-Lee (1999) *Weaving the Web*, Orion Business Books, London, UK

[6] J. Bradshow (1997) *Software Agents*, AAAI Press

[7] T. Bray *et al.* (eds.) (2000) *Extensible Markup Language (XML) 1.0 (Second Edition)*, World-Wide Web Consortium's Recommendation, Boston: `http://www.w3.org/TR/REC-xml`

[8] S. Buraga, S. Alboaie, A. Alboaie (2003) An XML/RDF-based Proposal to Exchange Information within a Multi-Agent System, in D. Grigoraş et al. (eds.), *Proceedings of NATO Advanced Research Workshop on Concurrent Information Processing and Computing*, IOS Press (to appear)

[9] S. Buraga (2001) *Web Technologies* (in Romanian), Matrix Rom, Bucureşti, Romania

[10] S. Buraga, G. Ciobanu (2001) *Programming Workshop in Computer Networks* (in Romanian), Polirom, Iaşi, Romania

[11] C. Callsen (1997) *Open Distributed Heterogeneous Computing*, PhD Thesis, University of Illinois at Urbana-Champaign

[12] J. Davies, D. Fensel, F. van Harmelen (eds.) (2003) *Towards the Semantic Web*, John Wiley & Sons

[13] D. Fallside (ed.) (2001) *XML Schema*, World-Wide Web Consortium's Recommendation, Boston: `http://www.w3.org/TR/xmlschema-0/`

[14] C. Gorman (2001) *Programming Web Services with SOAP*, O'Reilly and Associates

[15] D. Grigoraş et al. (2002) *MAIS – The Mobile Agents Information System Support for Creating Dynamic Clusters*, in Proceedings of ICA3PP, Beijing, China

[16] S. Haustein (2001) Semantic Web Languages: RDF vs. SOAP Serialization, *Proceedings of Semantic Web Workshop*, Hongkong, China

[17] E. Mangina (2003) *Review of Software Products for Multi-Agent Systems*, AgentLink.org: `http://www.agentlink.org`

[18] L. Moreau (2002) Agents for the Grid: A Comparison with Web Services (Part I: the transport layer), *IEEE International Symposium on Cluster Computing and the Grid Proceedings*, IEEE Press

[19] * * *, *Agentcitites Network*: `http://www.agentcities.net/`

[20] * * *, *Object Management Group Activity*: `http://www.omg.org/`

[21] * * *, *Semantic Web*: `http://www.semanticweb.org/`

[22] * * *, *SOAP Software*: `http://www.soapware.org/`

[23] * * *, *The Foundation of Intelligent Physical Agents* (*FIPA*): `http://www.fipa.org/`

[24] * * *, *Web Object Integration*: `http://www.objs.com/survey/web-object-integration.htm`

[25] * * *, *World Wide Consortium's Technical Reports*: `http://www.w3.org/TR/`

A Task-Oriented Compositional Mobile Agent Architecture for Knowledge Exchanges Between Agencies and Agents

Hong Zhou, Yufang Wang, Dia Ali and Maria Cobb
Department of Computer Science and Statistics
University of Southern Mississippi
Hattiesburg, Mississippi, USA 39406-5601
hong.zhou@usm.edu

Shahram Rahimi
Department of Computer Science
Southern Illinois University
Carbondale, Illinois, USA 62901
rahimi@cs.siu.edu

Keywords: Mobile Agent, Compositional, Task Oriented, Knowledge Exchange, AgentBee, Intelligence.

Received: August 7, 2003

This paper presents a task-oriented compositional mobile agent architecture named AgentBee. In this architecture, the mobile agent is in fact a task component. Each task component is recursively formed from sub-task components together with information defining the relationships among sub-task components and supplementing data. At the lowest level of this composition there exist primitive task components which are conceptually indivisible. Such a task-oriented compositional structure allows easy modifications of mobile agents. Thus, not only can it facilitate knowledge exchange between mobile agents and agencies, but also it can enhance the assembly of new mobile agents from existing components.

1 Introduction And Background

A mobile agent, because of its nature, has to deal with heterogeneous environments in which unexpected conditions may arise. This is the case for our geospatial data conflation project. One goal of our project aims at providing an automatic geospatial data conflation system in which geospatial data is stored in different databases at different locations and maintained by different organizations. The geospatial data exist in a variety of forms, such as different image files, VPF dataset, GML files, Oracle Spatial Information Management (SIM), etc. Mobile agents play an important role in this project in that they migrate among these databases searching for data conflicts and performing data conflation upon finding such conflicts. The challenge arises when the mobile agents have to deal with all the different geospatial data forms at different locations. Even more challengingly, new data forms may be introduced in at any times. One reasonable solution is to arm the mobile agent with all knowledge to deal with all possible conditions known so far. However, the following possible cases motivate us to search for another approach. First, one advantage of employing mobile agents in this project is to save networking bandwidth. Carrying all knowledge in a mobile agent would reduce this advantage. Second, since the databases are maintained by different organizations, some organizations may introduce new data forms or use new platforms that are unknown to the mobile agent. Third, different organizations may provide different computation libraries that only work well locally. In this case, the mobile agent needs to know how to use local facilities for better efficiency and quality.

The above challenge is in fact concerned with how the mobile agent accomplishes its tasks when unexpected conditions arise. It is obvious that this challenge could be answered if the mobile agent could update it itself by obtaining new functions from the agency for the unexpected conditions. For example, suppose that the mobile agent has two functions illustrated in Java as the following:

```
public void readImage(String fileName);
public void doConflation();
```

Suppose that the function *readImage* could only read in *tiff* and other image formats other than *jpeg*. If at one agency the given file is in *jpeg* format, then the *readImage* function fails there. However, once the mobile agent can obtain from the agency another *readImage* function that can read in *jpeg* files to replace/strengthen the existing *readImage* function, the problem is solved easily. In a similar manner, if the *doConflation* function of the mobile agent could not generate satisfactory results due to the new image content, the mobile agent could switch the *doConflation* function to another one obtained from the local agency if available.

It is noted that if the function *readImage* and *doConflation* are represented as objects in the mobile agent, they can be easily replaced with more powerful ones. In fact, both *readImage* and *doConflation* are two procedures that the mobile agent needs to execute to accomplish its goal and can be represented as two task components inside the mobile agent. This concept leads to our task-oriented compositional mobile agent architecture that is featured with high structural and functional flexibility, and adaptation.

Currently, most available mobile agent packages emphasize mobility, security, communication, and efficiency aspects of agents [11,12,13,14,15,16], while the learning ability, which is the major factor for flexibility and adaptation of the agents, is neglected. Nevertheless, there are few agent architectures in which the learning ability and knowledge exchange have been addressed. The compositional design strategy introduced by Brazier is one of these architectures that seem promising in enhancing knowledge exchange for multi-agent systems [1,2,3,4]. For more about Brazier's compositional agent design strategy, please refer to the references [1] and [2].

We employed Brazier's compositional design strategy to help analyze and construct our task-oriented compositional mobile agent architecture. Since the primary purpose of mobile agents is to perform a sequence of tasks on behalf of a client, we combine Brazier's process composition and knowledge composition to form a task composition approach such that there is only one fundamental component in our agent architecture: the task component. We named this mobile agent architecture AgentBee.

In AgentBee, each task component is recursively formed from sub-task components, together with information defining the relationships among them. At the lowest level of this composition, there exist primitive task components which are composed of some basic functions. Because the exchanges of task components can be achieved easily, such a task-oriented compositional structure allows easy modifications and adjustments for mobile agents, hereby facilitating knowledge exchange between agents and agencies.

The rest of the paper is organized as the following. Section 2 formally defines the AgentBee architecture and its primary components. Section 3 describes the knowledge exchange processes between agencies and agents. Section 4 illustrates how AgentBee can be used in our project. In section 5 we discuss the advantages and disadvantages of AgentBee. Finally, in section 6 we draw a short conclusion on AgentBee.

2 AgentBee Architecture

Considering the fact that mobile agents have to perform a set of tasks, AgentBee defines a mobile agent to be a task component that is composed of a group of sub-task components. Each task component, other than *primitive* task components, is constructed recursively from an assembly of sub-task components together with information defining the relationships among sub-task components and supplementary data. Each task component is defined individually with its own functionalities, inputs, outputs, and the relationships among its sub-task components. A *primitive* task component is not composed of any other components, but is composed of some basic information and functions.

In this section, we formally define task components, mobile agents, and agency's knowledge storage in the AgentBee architecture.

2.1 Task Component

Task components are the fundamental components in the proposed architecture. A specific task component is defined to be an independent working unit that can accomplish a specific function given its required input is supplemented. A task component T is composed of multiple sub-task components and can be expressed as:

$$T = \{<T_i>, <I_j>, C, D\} \mid T^0$$
$$T^0 = \{P, D\}$$

- $<T_i>$ represents a set of n sub-task components inside the parent task component T, where $n > 0$, and $i = 1,2,\ldots,n$. Each T_i has a name that

uniquely identifies it inside *T*. $<T_i>$ is the function component of *T*.

- $<I_j>$ represents a set of *m Information Objects* which provide the sub-task components with data to use and rules to obey. This set may or may not be empty (*m >= 0)*. There are two major categories of *Information Objects*. One category provides rules (*Rule Information Object*), and the other provides supplemental data (*Data Information Object*). $<I_j>$ is the data component of *T*.
- *C* represents the *Control Object*. *C* is the flow control component in *T*. A control object can be viewed as a table holding the following information: 1) the executing order of the sub-task components, 2) the number of times each sub-task component should be executed at its turn, and 3) the identity of the required *Information Object* for each sub-task component. When *C* does not specify the identities of the *Information Objects* for a sub-task component T_i, the *Information Objects* required by T_i become the input requirements of *T*. The following example table helps explain the structure of *C*.

Table 1. Sample structure of a Control Object

Ti	*Execution Times*	*Information Objects*
T_1	*1*	I_{11} ,I_{12}
T_2	*1*	I_{21}
T_1	*1*	I_{11} ,I_{12}
T_3	*2*	

Table 1 gives an example of control object *C*. In Table 1, task component *T* has three sub-task components T_1, T_2, and T_3. The execution order of the sub-tasks is T_1, T_2, T_1, T_3 and they are separately executed in that order 1, 1, 1, and 2 times. The required in*formation Objects* for T_1 are I_{11} and I_{12}, and for T_2 is I_{21}. *C* does not link any *Information Objects* for T_3. Suppose T_3 does require an inform*ation Object* such as I_{31} which does not exist in *T*, then I_{31} becomes the input requirement of *T*, i.e. *T* requires an *Information Object* as input to supply T_3.

- *D* represents the *domain* component in *T* that at least distinguishes the task component's functionality, required input (*Information Objects*), and output. *D* is primarily composed of D_p and D_s. D_p represents the primary domain that defines the task component's functionality, required input, and output. D_s represents the sub-domain that declares the implementation level of this task component. Here, "implementation level" is a user-defined property that specifies the efficiency (and/or quality) of the task component. Besides D_p and D_s, *D* could contain other describing information about the task component. For example, to platform dependent task components, *D* can contain information to specify on what platforms this task component could generate quality result. D_p is mandated to exist to define the domain. Nevertheless, the existence of D_s is optional but could greatly help in distinguishing the task component. If two task components have the same primary domain D_p but different D_s, then the two task components must be able to accomplish the same function with the same input, but in different ways and probably with different efficiencies and/or qualities. Therefore D_s could be used to help pick the best candidate from multiple task components all of which have the same D_p. Since *T's* functionality is defined by $<T_i>$ in order, one way to determine the primary domain *D* of *T* is based on the primary domains of sub-task components $<T_i>$ that are organized in a specific order. For example, given the *T* and *C* in Table 1, suppose T_1, T_2, and T_3 have domain D_1, D_2, and D_3. Then *T's* primary domain D_p is defined as $D_{1p}D_{2p}D_{1p}D_{3p}^2$, where D_{3p}^2 corresponds to the fact that T_3 is executed twice.
- The vertical bar | represents the disjunctive relationship *OR* such that *T* may or may not be a primitive task component.
- T^0 represents a *primitive* task component. A *primitive* task component is conceptually atomic, i.e. indivisible. It is designed to achieve only one simple function.
- *P* represents the specific function of the primitive task. It is the function component in T^0.

The above notation states that a task component either is composed of a set of other task components, information objects, and domain information, or is just a primitive task component. It is important to observe that the above definition is very basic. It only depicts the foundation of AgentBee, but leaves great space for further expansions. The following expression further defines the relationship between the sub-task components and the *Information Objects* inside *T*.

$$S \leftarrow T_i + [I_j],$$

states that the execution of a sub-ask component T_i with input $[I_j]$ ($[I_j]$ represents zero or more *Information Objects*) generates result *S*. It is important here to note that $[I_j]$ may be from the *Information Object* subset $<I_j>$ that is in the same task component *T* as T_i is, or it is dynamically supplemented to T_i.

Inside the parent task component *T*, each of the sub-task components can work independently from each other by definition. However, without proper organization, the final result of the parent task *T* is unpredictable. Thus the control object C becomes extremely important because it is *C* that organizes the sub-task components in order to accomplish *T*'s functionalities. The same set of sub-task components can be organized in different ways to make *T* behave totally different. Also, with different *Information Objects*, the sub-task components can achieve different results. Thus, $<T_i>$, $<I_j>$, and *C* all are important for *T* to function properly.

2.2 Mobile Agents' Structure

Mobile agent *A*, in this architecture, is treated as a task component with an extra mobility sub-task component in its sub-task set. This means that mobile agent *A* includes a specific sub-task component which is dedicated to the mobility of the agent:

$$A = \{<T_i>, <I_j>, C, D\}.$$

Thus, a mobile agent is basically a task component with the ability to migrate from one agency to another. For the mobile agent to migrate to multiple agencies, the sub-task component for mobility appears multiple times in the *Control Object* and each time with a different *Information Object* linked.

2.3 Agency's Knowledge Storage

In AgentBee, each agency includes a knowledge storage. In this paper, we only discuss the architecture of the agency's knowledge storage, and skip its rest parts. The knowledge storage *KB* of an agency is composed of two layers and can be expressed as:

$$KB = R(\{<T>^m\})$$

- $\{<T>^m\}$ represents the bottom layer that is composed of a bank of task components. It represents the knowledge contents of *KB*. The size of the knowledge storage depends on the size of the bank of task components. Since agency does not move, the bank of task components could be fairly large. Besides, there must be some redundancy in $\{<T>^m\}$. Here, "redundancy" means that the same task component appears as an independent component in $\{<T>^m\}$ and at the same time exists in another task component as a sub-task component. For example, T_a exists as a sub-task component of T_A. Meanwhile, both T_a and T_A co-exist independently in $\{<T>^m\}$.
- *R* represents the top reasoning and interface layer that communicates with the mobile agents and is powered with reasoning abilities to analyze and process agent requests (*AR*). The agent request (*AR*) is sent to *R* by the mobile agent and contains information about mobile agent task components and the reason for the request. *R* is expressed as a function which uses the lower layer of *KB* and *AR* to produce its response. The response is generated in the form of knowledge exchange. For example, suppose the *AR* indicates that the request is to obtain a task component able to read *jpeg* image files for agent *A*, if the *KB* has such a task component, then *R* produces the response that is providing the task component to agent *A*. The complexity of *KB* lies in *R*. *R* could be fairly complex to include an expert system or could be fairly simple to only provide task components in response to the agent request.

3 Knowledge Exchange Scenarios

Since one major advantage of mobile agents is to save networking bandwidth, the size of the knowledge that a mobile agent carries should be optimized. Hence in AgentBee, mobile agents only carry small amounts of knowledge, while using agencies' knowledge facilities to perform intelligent actions. They carry only the minimum required knowledge as their local knowledge contents when they first start. The agency knowledge storages are the primary knowledge resources. Since mobile agents work inside the agency environments, we define that knowledge exchanges in AgentBee only happen between the agency and the mobile agent. There is no direct knowledge exchanges among mobile agents. This definition may be too restrict because direct agent-agent communications can happen, however such a definition does help simplify the construction of AgentBee at this time.

Structural flexibility leads to functional flexibility. The compositional and task-oriented structure allows

the mobile agent's contents to be modified easily. A component in the mobile agent could be switched in and out. This switch-in and switch-out process is in fact a knowledge exchange process in AgentBee. In AgentBee, two basic types of knowledge exchanges are defined. They are information exchange and function exchange.

3.1 Information Exchange

I_j in T can be modified to change the state of a task component. This process is called information exchange in AgentBee and can be expressed as:

$$S' \leftarrow T_i + [I_j'].$$

The above expression states after the modification on I_j so that I_j becomes I_j', the original result that T_i achieves is changed (in some cases however, $S' = S$, i.e not changed). It is important to point out here that the information exchange does not have any impact on the domain value D of T.

3.2 Function Exchange

Changes to $<T_i>$ alter the functional capability of T, and can be expressed as:

$$T_j^i \leftarrow E(T_i, T_j), \text{ when } D_j = D_i.$$
$$T_i(T_j) \leftarrow E'(T_i, T_j), \text{ when } D_j = D_i.$$

- $T_j^i \leftarrow E(T_i, T_j)$ represents a task component replacement process in which T_i is replaced with T_j. T_j^i states that T_j replaces T_i to fulfill the same task as T_i, but maybe in a different way.
- $T_i(T_j) \leftarrow E'(T_i, T_j)$ represents a task component supplementation process in which T_j is bounded with T_i so that T_i now has the ability of T_j besides its original ability.
- $D_j = D_i$ states the *domain rule*. D_j represents the domain of T_j and D_i represents the domain of T_i. *Domain rule* requires that only when the primary domain of D_j is the same as the primary domain of D_i, i.e. T_j can achieve what T_i can with the same input but in a different way, $E(T_i, T_j)$ and $E'(T_i, T_j)$ can happen. *Domain rule* guarantees that after function exchange between T_i and T_j, the original functional goal of T_i is not changed, though how it is achieved may be different.

Since every component is composed of sub-components, the function exchanges are actually simple plug-in and/or plug-out procedures. A simple similar view of this process is the part exchange of a car. When the front tires of a car wear out, we change the front tires with new tires. In a similar way, when a sub-task component is incompetent, we change this sub-task component. However, the organization of the sub-task components is kept the same. Because the higher level components are usually much more complicated than the lower level ones and therefore finding matched components in high levels may be difficult, it is useful to keep the function exchanges processed at as low a level as possible. The component-based architecture ensures that function exchanges between two task components T_1 and T_2 can always be completed at a lower level as long as the function exchanges on each sub-task component of T_1 and T_2 can be performed. Thus with proper design, all function exchanges can be eventually processed at the level of *primitive* task components. Also, the compositional structure allows easy identification of the sub-task component(s) that require exchanging, and this process can also be traced to one or more *primitive* task components.

4 Project Case Study

In this section, we discuss how we can employ the AgentBee architecture to solve some critical issues in our data conflation project. As discussed in section 2, AgentBee structure is flexible for further expansions. In the case of our project, we expand the AgentBee structure to provide customized mobile agent system. Also, only the function exchange is discussed in the project case study.

4.1 Long-Term/Short-Term Knowledge

In its lifetime, the mobile agent continues to learn. Some knowledge obtained may last for a long time or even the lifetime of the mobile agent, while some may be temporary. For example, some knowledge is necessary only for working at a specific agency. Thus, such knowledge should be temporary for the mobile agent and may be discarded right before it leaves this agency. Two types of knowledge based on how long they last are introduced into AgentBee. One type is long-term knowledge, and another is short-term knowledge. Since knowledge is interpreted in the form of task components, by simply introducing D_t (stands for the type of the task component T) in D, each task component is easily marked as either long-term or short-term.

In our geospatial data conflation project, solving system equation is a commonly used routine. When the mobile agent finds any data conflicts, it performs a set of processes including solving system equations to solve the conflicts. The mobile agent has a built-in task component (T_s) for solving system equation. T_s is developed to work with different platforms and

therefore it can not achieve the highest efficiency at all agencies. The databases are distributed at different locations and on computers of different platforms. On one Unix computer, the very efficient library ATLAS [17] for solving system equation is installed. Now an agency specific task component T_s' is developed on this agency using this efficient library to solve system equations. Since T_s has great impact on the conflation efficiency, this agency demands that every mobile agent has to use the T_s' to solve system equations if the implementation level (D_s) of the T_s of the mobile agent is lower than the implementation level of T_s'. Thus, function exchanges happen at this agency. T_s' is short-term knowledge because it only works on this specific agency. Hereby, the mobile agent discards T_s' before it leaves this agency and continues to use its own T_s at next agency.

4.2 New Data Forms

Mobile agents work in a distributed environment in which each database and agency may be maintained by different organizations. Each organization adds new data, modifies existing data, and develops new algorithms to improve the conflation performance and quality. Currently, there are various ways to store geospatial data, such as VPF, GML, Oracle SIM, etc. It is possible that each agency stores data in its own preferred way. Also, one type of data used in our project is image. The images that are created and/or collected by each agency may be in different formats. Thus it becomes difficult for the mobile agents to be aware of all the possible data types beforehand. Therefore, each agency is responsible to develop new conflation task components for the new data type or forms it introduces in. Whenever a mobile agent meets an unknown data type at an agency, it demands the agency for a proper conflation task component. Here again, function exchanges happen. If the newly obtained task component is not agency dependent, the mobile agent carries this task component with it and updates each other agency with this component. In this sense, the mobile agent spreads out the new knowledge among agencies.

4.3 Automatic Agent Creation

In our project, mobile agents play the most critical role. They travel among various agencies to update the databases automatically. Frequently, mobile agents for new tasks (hereby with new functions) are required to be generated. Thus, an automatic agent creation facility is a big plus to our project. The compositional architecture enhances the automatic creation of mobile agents. Once the user knows what the mobile agent will do, the user can browse the list of the task components of the agency and collect some of them based on their specifications. This collection is then organized hierarchically to assemble the desired mobile agent. There are 3 basic steps for automatic agent creation.

- Step 1 is to specify the tasks of the mobile agent. This is a top-down task decomposition process. In this process, each task of the mobile agent is decomposed into a sequence of multiple sub-tasks, and then each sub-task could be further decomposed into a sequence of sub-sub-tasks. This decomposition process continues until the user decides that there is no need for further decompositions. Thus, a hierarchy tree of task specifications is built up and the root is the mobile agent itself.
- Step 2 is to specify the required data and rules for each task in the hierarchy tree.
- Step 3 is a down-top process. Starting from the bottom of the hierarchy tree built in step 1, the user finds a corresponding task component in the agency for each leave of the tree. Also, based on the date/rules specified in step 2 for each leave, the user adjusts the existing *Information Objects* in the task components or creates new ones for them. These bottom level task components are then organized to construct the one-level higher task components of which *Information Objects* are again adjusted or newly created. Furthermore, these "one-level higher" task components are organized to construct other task components that are of even higher levels. Such a construction process proceeds until the mobile agent is constructed.

In fact, this mobile agent assembly process not only helps the creation of new mobile agents, but also helps secure the system. If users of different system privileges could only browse specific groups of task components in the agency to create mobile agents, a user with limited privileges could only create mobile agents of limited functions.

5 Discussion

While mobile agent technology is drawing more and more attentions because of its great potentials in various applications, the intelligence issue of mobile agent technology begins to show its importance in the success of this technology [5,6,7,8,9]. However, so far the research works on the intelligence issue of mobile agent technology is far away from enough to properly show its importance. To our best knowledge, there still exists no mobile agent design

that is specifically for the mobile agents to become more intelligent. Meanwhile, our geospatial data conflation project needs an intelligent mobile agent system that could deal with unexpected conditions. Thus, we propose the task-oriented compositional mobile agent architecture AgentBee that can facilitate knowledge exchanges between mobile agencies and agents.

In AgentBee, a task component T (except T^0) is composed of four components: *Control Object C*, *Domain D*, a set of *Information Objects*, and a set of sub-task components. Each sub-task component can fulfill a specific task, but they have to be organized in order to accomplish the parent component's task. This organization is dependent on the *Control Object C*. Thus, *C* is the central control of *T*. In the process of knowledge exchange, since the identity of *T* must not be changed, *D* is therefore unchangeable. In the current AgentBee architecture, *C* is also not changeable because changes to *C* may change what *T* can achieve and therefore change *D*. Only the *Information Object* set and the sub-task component set of *T* can be modified or exchanged.

The task-oriented compositional structure allows the function exchanges to be as simple as plug-in/plug-out processes. This is the advantage of this compositional mobile agent architecture. In the information exchanges, the modification of any *Information Objects* is a much more complicated process. An *Information Object (I)* contains data or rules. There exists uncertainty in what data/rule in *I* should be modified and how to make the modifications. To identify what data/rule in *I* should be modified, the agent request *AR* has to be very definite such that the reasoning layer *R* in the agency knowledge storage can make the identification. How to modify the *Information Object* is a decision of *R*. Thus, there exists a great amount of reasoning intelligence in *R*. Due to the space limitation, we do not discuss *R* in detail in this paper. Nevertheless, there exist restrictions for modifications of *Information Objects* so that such modifications will never change the domain of the task component.

We defined that function exchanges can happen only when the *domain rule* is met, i.e. the two involved task components have the same D_p value. However, it is possible that the mobile agent ends up obtaining an unwanted and inferior component by the function exchange. Thus, it is important to differentiate task components of the same D_p value if they have different capabilities. For example, D_r can be added into D to represent the RANK value of each task component. Higher D_r stands for more advanced capability. So, with the introduction of D_r, we could extend domain rule such that not only must the two involved task components have the same D_p value, but also the task component from the agency should have higher D_r value compared to the task component from the mobile agent.

Knowledge sharing and exchanges among agents are necessary for intelligent mobile agents. For simplicity, knowledge exchanges are limited only between agencies and agents in AgentBee. The knowledge exchanges among agents can be realized based on the same *domain* rule if the agents themselves can have the intelligences to analyze the agent requests. However, it is an overhead for mobile agents to carry the intelligence for analyzing agent requests.

Even though the compositional architecture of AgentBee has various advantages, it does impose some negative effects on the mobile agent system design. As the mobile agent technology becomes more and more popular, eventually there will be one or more standards adopted for this technology. Currently, two technology standards proposed for mobile agents, MASIF and FIPA, are gradually accepted [10,18,19]. However, none of the two standards address the knowledge exchange issue. Meanwhile, AgentBee is specifically designed for structural flexibilities and knowledge exchanges, it does not accommodate to either standard.

As the mobile agent obtains more and more knowledge, its size becomes larger and larger. This impacts the transportation efficiency of the mobile agent. Some of the knowledge the mobile agent is carrying becomes useless and should be discarded. Also, when the mobile agent carries multiple task components for the same functionality, it has to select one from them to achieve this functionality at every agency. This decision-making process could degenerate the performance of the mobile agent.

6 Conclusion

Intelligence in mobile agents is a neglected but important issue. Nevertheless, it is relatively difficult to realize the intelligence in mobile agents. One reason is that mobile agents are continuously traveling in different environments; their structures have to be flexible and subject to modifications for knowledge acquisitions. AgentBee is compositional in that its basic components are exchangeable, i.e. subject to modifications. This greatly enhances the

knowledge exchanges between the agents and the agency, especially and specifically the function exchanges.

7 Acknowledgement

This work is partially supported by the project "Intelligent Database Agents for Geospatial Knowledge Integration and Management" which is funded by the NIMA, University Research Initiative Award # NMA201-00-1-2004.

8 References

[1] F.M.T.Brazier, C.M. Jonker, J. Treur, N.J.E. Wijngaards, "Compositional Design of a Generic Design Agent," *Design Studies Journal*, 22:439-471, 2001.
[2] F.M.T.Brazier, C.M. Jonker, J. Treur, "Principles of Compositional Multi-Agent System Development," *Proceedings of the IFIP'98 Conference IT & KNOWS*, 1998.
[3] F.M.T.Brazier, B.D. Keplicz, N.R. Jennings, J. Treur, "Formal Specification of Multi-Agent Systems: a Real-World Case," *Proceedings of the First International Conference on Multi-Agent Systems, ICMAS'95*, MIT Press, pp 25-32, 1995.
[4] F.M.T.Brazier, N.J.E.Wijngaards, "Automated (Re-)Design of Software Agents," *Proceedings of the Artificial Intelligence in Design Conference 2002*, Kluwer Academic Publishers, pp 503-520, 2002.
[5] D. Kotz, R. S. Gray, "Mobile Agents and the Future of the Internet," *ACM Operating Systems Review*, 33(3):7-13, August 1999.
[6] Danny B. Lange and Mitsuru Oshima, "Seven Good Reasons For Mobile Agents," *Communications of the ACM*, 42(3):88-89, March 1999.
[7] T. Magedanz, K. Rothermel, S. Krause, "Intelligent Agents: An Emerging Technology for Next Generation Telecommunications?"
Proceedings of IEEE INFOCOM ´96, pp. 464-472, San Francisco, USA, March 24-28, 1996
[8] T. Magedanz, R. Popescu-Zeletin, "Towards Intelligence on Demand - On the Impacts of Intelligent Agents on IN," *Proceedings of 4th International Conference on Intelligent Networks (ICIN)*, pp. 30-35, Bordeaux, France, December 2-5, 1996.
[9] M. Breugst, T. Magedanz: "Mobile Agents - Enabling Technology for Active Intelligent Network Implementation," *IEEE Network Magazine*, 12(3):53-60, Special Issue on Active and Programmable Networks, August 1998.
[10] P.K.Menelaos, F.G.Chatzipapadopoulos, I.S. Veieris, G. Marino, "Mobile Agent Standards and Available Platforms," *Computer Networks*, 31:1999-2016, 1999.
[11] http://mole.informatik.uni-stuttgart.de/mal/preview/preview.html.
[12] http://dsonline.computer.org/agents/agentsprojects.htm.
[13] http://www.ikv.de/products/grasshopper.
[14] http://www.trl.ibm.co.jp/aglets.
[15] http://www.objectspace.com/voyager.
[16] http://www.merl.com/projects/concordia.
[17] http://math-atlas.sourceforge.net/.
[18] http://www.fipa.org.
[19] ftp://ftp.omg.org/pub/docs/orbos/98-03-09.pdf.

Towards a Modelling Methodology for Fault-Tolerant Multi-Agent Systems

Sehl Mellouli, Bernard Moulin and Guy W. Mineau
Department of Computer Science and Software Engineering,
Laval University, Quebec, G1K 7P4 Canada
Email:{sehl.mellouli, bernard.moulin, guy.mineau}@ift.ulaval.ca

Keywords: methodology, multi-agent systems, design

Received date: August 18, 2003

Multi-Agent Systems (MAS) can be applied to a wide range of applications such as organizational, Internet, adaptive, or FIPA compliant applications. Developing such systems requires agent software engineering methodologies. To this end, many methodologies have been proposed such as Gaia (for organizational or Internet applications), ADELF (for adaptive applications) or SABPO (for FIPA compliant applications). However, no methodology has been proposed to deal with fault-tolerant MAS. In fact, agents are prone to failures, and thus are MASs. So, the MAS may not reach its objectives in case of agent failure. Hence, it is important to check a MAS design in order to prevent agent failures, so that we reduce failure risk at run time. The aim of this paper is to present an overview of agent oriented software engineering methodologies and to propose some guidelines towards a modelling methodology for fault-tolerant MAS.

1 Introduction

Most Multi-Agent Systems (MASs) operate in an environment in which many troublesome situations might occur during execution. Some of these situations could lead agents to fail. Meanwhile the MAS must continue to operate despite this failure in order to achieve its tasks. So, the MAS has to be tolerant to agent failure, and thus must be fault tolerant. This is what is called a Fault-tolerant Multi-Agent system.

Developing a MAS requires software engineering methodologies. We can refer to object oriented methodologies such as UML [20] or agent oriented methodologies such as GAIA [22], ADELF [1] or SABPO [4]. They propose, in general, common phases that are analysis and design phase, and use common concepts such as role, autonomy, and communication. Since we aim at studying agent methodologies in order to define a modeling methodology that wil be used to build fault-tolerant MAS, we propose to study nine methodologies and identify the modeling phases that are relevant to our goal.

Knowing that four main sources of faults are identified when developing a software system [10]:

- Inadequate software specification
- Software design error
- Processor failure
- Communication error

Our intended methodology deals with agents faults at design level, so that the two first main sources of faults are addressed as will be discussed in Section 3.

This paper is organized as follows. In section 2, we present an overview of agent-oriented software engineering methodologies. In section 3, we propose guidelines towards a modelling methodology for faut-tolerant multi-agent systems. Section 4 concludes.

2 Overview of agent-oriented software engineering methodologies

Multi-agent systems can be applied to solve problems in various domains such as organizational [22], FIPA complianr applications [4], and Internet related domains [24]. To this end, several methodologies have been defined to develop MAS, such as GAIA [22], SABPO [4], and ADELF [1]. We found that most of the studied methodologies have similar phases or use similar modelling techniques. So we present in this section an overview of nine agent-oriented software engineering methodologies: MAS-CommonKADS, GAIA, extension of GAIA for Internet applications, SODA, AALAADIN, Adelf, SABPO, MESSAGE/UML and Tropos, in order to extract commonalities between them so that we can propose some guidelines towards a modelling methodology for fault-tolerant MAS (Section 3).

2.1 The MAS-CommonKADS methodology

MAS-CommonKADS [11] is a Multi-Agent System design methodology based on the CommonKADS methodology [19]. It has three phases: the conceptualization phase, the analysis phase, and the design phase.

2.1.1 The conceptualization phase

The conceptualization phase helps developers to understand the problem to be solved. The main outputs of this phase are two models: use cases (based on actors), and MSC (Message Sequence Charts) used to describe interactions between different actors.

2.1.2 The analysis phase

The second phase is analysis. It carries out a requirement specification of the MAS through the development of five models: the agent model, the task model, the coordination model, the knowledge model, and the organization model.

1. The Agent model consists of identifying types of agents, describing them and determining their instances.

2. The task model consists of a task decomposition, goals determination, and the identification of tasks ingredients.

3. The coordination model consists of describing the interactions and coordination protocols between agents. It shows the dynamic relationships between them.

4. The knowledge model is carried out by the expertise model. This model consists of determining the application knowledge model and the problem solving knowledge. The application knowledge model consists of the domain knowledge, the inference knowledge and the task knowledge. The domain knowledge represents the declarative knowledge of the problem modelled as concepts, properties, expressions and relationships using OMT [18]. The inference knowledge represents the inference steps performed to solve a task. The task knowledge represents the order of the inference structures. The problem solving knowledge specifies how the inference is carried out.

5. The organization model represents the organization in which the MAS will be deployed and the software organization of the MAS. It shows the static or structural relationships between the agents. It is based on the OMT notation (aggregation, inheritance). This model is the specification of the structural relationships between human and/or software agents, and the relationship with the environment.

2.1.3 The design phase

The third phase is design. It carries out the design model. This model consists of the agent network design, the agent design and the platform design.

1. The agent network design determines the infrastructure of the MAS-system according to the network, knowledge and coordination facilities. The network facilities are for example the agent name service, the registering and subscription service, the security level, the encryption and authentification, the transport/application protocol and the accounting service. The knowledge facilities are the ontology server and the knowledge representation language translators. The coordination facilities are the available coordination protocols, the protocol servers and the group management facilities.

2. The agent design consists of agents where each agent is subdivided in modules for user-communication, agent communication (inferred from the coordination model), deliberation and reaction (from the expertise, agent and organization models), and external skills and services (from the agent, expertise and task models).

3. The platform design is the selection of the needed software and hardware to implement the MAS.

2.2 The Gaia methodology

The Gaia [22] methodology is applicable to a wide range of multi-agent systems where agents are cooperative and in which the system is closed. It is composed of two main phases: the analysis phase and the design phase. These two phases are preceded by the requirement statement.

2.2.1 The analysis phase

The objective of the analysis phase is to understand the system and its structure. To this end, two models are proposed: the role, and the interaction models. In the role model, an organization is seen as a collection of roles that stand in certain relationships to one another. The interaction model presents the links between roles. It consists of a set of protocol definitions, one for each type of inter-role interaction.

2.2.2 The design phase

The objective of the design phase is to transform the analysis models into a sufficiently low abstraction levels in order to implement the MAS. This phase generates three models that are the agent model, the services model, and the acquaintance model.

- The agent model identifies the agent types and instances. It is defined using a simple agent type tree, in which leaf nodes correspond to roles and other nodes correspond to agent types. The agent instances that will appear in a system are documented by annotating agent types in the agent model.

- The services model identifies the main services that are required to achieve the agent role. It specifies the main properties of these services. A service corresponds to an agent's function.

- The acquaintance model defines the communication links that exist between agents. They do not define which messages are sent or when they are sent. They only indicate that communication pathways exist. It is a graph whose nodes correspond to agent types and arcs correspond to communication pathways.

2.3 Extension of Gaia for Internet Applications

Gaia is suited for closed systems. Somehow, a wide range of MAS applications are used in open environments such as the Internet. Gaia has been later adapted to deal with open environments. In [24], it is proposed to use coordination model to apply Gaia to Internet applications.

In Gaia, the organization structure of the system is static; neither the number of agents nor their inter-agent relationships change at run time [24]. The agents globally exhibit cooperative behavior; they have a global goal and do not exhibit competitive or self-interested behaviors. A coordination model is exploited in the context of designing MAS to be used on the Internet. It makes it possible to enact social laws in the system and to control the execution of foreign and self-interested agents. It provides a formal framework in which the interaction of a set of concurrent activities can be expressed. It consists of three elements: the coordinables (the agents), the coordination media (semaphores, monitors, channels, blackboards, etc.) and the coordination laws (communication language, and coordination language that is a set of interaction primitives and their semantics).

The coordination models can be divided into two categories: a data-driven category in which agents interact with the external world by exchanging data structures through the coordination media, which basically acts as a shared data space, and a control-driven category in which coordination is done via well-defined input/output ports. The presence of the coordination media implies that it is no longer necessary to determine all the interaction links between all the possible agents that can access to the application and to define all the possible interaction protocols. An agent can be unbound so that it doesn't know with which agent it will interact. The coordination media can be programmed so that the interaction between two agents becomes dynamic. It is also possible to constrain the behavior of the agents during their interactions, or to monitor and control all the interactions between agents. The coordination media enforces whatever social laws have to be respected by the agent system in order to carry out the social tasks.

By adding the coordination media to Gaia, the expected output of the analysis phase are well-defined role and interaction models, in addition to a well-defined model of social laws. The expected output of the design phase are an agent and a service models detailed enough in order to implement the agents. In addition, the behavior of the coordination media should make it possible to implement the coordination media, whatever media is actually exploited.

2.4 SODA

The SODA [17] methodology is suited for internet-based systems. It consists of two phases, the analysis phase and the design phase. During the analysis phase, the application domain is studied and modelled, the available resources and the technological constraints are listed, the fundamental application goals and targets are pointed out. The design phase deals with the representation of the abstract models obtained during the analysis phase.

2.4.1 Analysis phase

The analysis phase generates three models that are: the role model in which the application goals are modelled in terms of the tasks to be achieved, the resource model in which the application environment is modelled in terms of the services available, and the interaction model in which the interactions involving roles are represented.

In the role model, the tasks are expressed in terms of the responsibilities they involve, the competencies they require, and the resources they depend upon. The tasks are classified as either individual or social ones. Each individual task is associated with an individual role. A role is defined in terms of responsibilities. Social tasks are assigned to groups. Groups are defined in terms of both the responsibility related to their social task and the social roles participating in the group. A social role describes the role played by an individual within a group.

In the resource model, the services express functionalities provided by the agent environment to the multi-agent system such as querying a sensor and verifying an identity. Each service is associated with an abstract resource, which is firstly defined in terms of the services it provides. Each resource defines abstract access modes (permissions), modelling the different ways in which the corresponding service can be exploited by agents.

The interaction model presents interactions involving roles, groups and resources in terms of interaction protocols. An interaction protocol associated with a role is defined in terms of the information required and provided by the role in order to accomplish its individual task. An interaction protocol associated with a resource is defined in terms of the information required to invoke the service provided by the resource itself, and by the information returned when the invoked service has been brought to an end, either successfully or not. An interaction protocol associated with a group governs the interactions among social roles and resources in order to enable the group to accomplish its social tasks.

2.4.2 The design phase

The design phase enable the designer to create three models: the agent model in which individual and social roles are mapped upon agent classes, the society model in which groups are mapped onto societies of agents, and the environment model in which resources are mapped onto infrastructure classes. In the agent model, an agent class is defined as a set of one or several roles. It is characterized by the tasks, the set of permissions, and the interaction protocols associated with its roles. In the society model, each group is mapped into a society of agents. An agent society is characterized by the social tasks, the set of permissions, the participating social roles, and the interaction rules associated with its groups. In the environment model, resources are mapped onto infrastructure classes.

2.5 The AALAADIN methodology

AALAADIN [6] is a generic meta-model for multi-agent systems. The core concepts of AALAADIN are roles and groups. A group is defined as a set of agents. A role is defined as an abstract representation of an agent function, a service or an identification within a group. In AALAADIN, the agents are defined by their functions in an organization, that is by their roles and the set of constraints which they must accept in order to be able to play these roles. Agents can play different roles in different groups. AALAADIN's methodological approach consists in determining first the group structure by identifying all the roles and interactions that can appear within a group, and second the MAS organizational structure, that is the set of group structures expressing the design of a multi-agent organization scheme.

2.6 ADELF, a methodology for adaptive multi-agent systems engineering

ADELF [1] is suited for adaptive multi-agent systems in which the environment is unpredictable and the system is open. A strong adaptation is the ability that the system must possess in order to take into account unpredictable events and to react to evolutionary environments. In adaptive multi-agent systems, the agents are involved in cooperative interactions. ADELF proposes three workflows: the requirements workflow, the analysis workflow and the design workflow.

2.6.1 The requirements workflow

In the requirements workflow, ADELF provides a model composed of the target system (by a set of keywords), and the system environment. This workflow focuses on what may be in interaction with the studied system in terms of passive or active entities or constraints. It requires a characterization of data flows and interactions between passive or active entities and the system. These interactions are expressed by collaboration and sequence diagrams [20].

2.6.2 The analysis workflow

In the analysis workflow, ADELF proposes to first identify the agents by performing a domain analysis to produce a preliminary class diagram. Each agent has to be analyzed as a system. Second, it proposes to study the interactions between the different entities as a set of sequence diagrams (like in AUML [15]) and activity diagrams which explain the possible interactions between the different entities within the system at each level.

2.6.3 The design workflow

In the design workflow, ADELF defines the agent model and the Non Cooperative Situations model (which could be related to exceptions in classical programs). The agent model represents the relationships between agents. The non cooperative situation model deals with the non cooperative situations that are situations in which the multi-agent system cannot reach its objectives. In addition, the design phase produces the architecture of the system in terms of blocks, classes, agents and interactions.

2.7 SABPO: A Standard Based and Pattern Oriented Multi-Agent Development Methodology

A MAS behaves like a social organization in which each agent plays a specific role. The FIPA standards define the required services to construct MASs working in open environments and define interaction patterns in order to build robust organizational structures. Any attempt of methodology development should take the FIPA standards as a basis [4]. In FIPA based agent systems, agent interactions are specified using the pre-defined FIPA interaction protocols. SABPO [4] tries to identify required interaction protocols based on the system requirements during the analysis phase. The approach is composed of an analysis phase and a design phase.

2.7.1 The analysis phase

In the analysis phase, the following models are developed: the role model and the interaction model. The role model identifies the roles and responsibilities of these roles in order to satisfy the organization's global goals. SABPO introduces two roles that comply with the FIPA abstract architecture. These roles are 'Directory Service Provider' and 'Ontology Service Provider'. The interaction model defines the interaction protocols between agents. These interactions are documented using AUML [15].

2.7.2 The design phase

In the design phase, three models are developed: the ontology model, the agent model and the detailed interaction model. The ontology model extends the ontology knowledge obtained during the analysis phase.

The agent model defines the agent types and assigns the roles defined in the analysis phase to the agent types. The detailed interaction model maps the interaction protocols identified in the analysis phase to the FIPA specifications.

2.8 Agent oriented Analysis using Message/UML

MESSAGE [3] stands for: Methodology for Engineering Systems of Software AGEnts. It proposes five model views that are: the organization view, the goal/task view, the agent/role view, the interaction view and the domain view. The organization view (OV) shows concrete entities (agents, organizations, roles, resources) of the system and its environment, and coarse grained relationships between them (aggregation, power, and acquaintance relationships). The goal/task view (GTV) shows goals, tasks, situations and dependencies between them. The agent/role view (ARV) focuses on the individual agents and roles. In the interaction view (IV) a designer must, for each interaction between agents/roles, show the initiator, the collaborators, the motivator, the relevant information supplied/achieved by each participant, the events that trigger the interaction, and other relevant effects of the interaction. The domain view (DV) shows the domain specific concepts and relations that are relevant to the system under development.

The analysis process is based on a refinement approach. The system is viewed as a set of organizations that interact with resources, actors, or other organizations. Actors may be human users or other existing agents. The modelling process starts by building the organization and the goal/task views. These views act as inputs to creating agent/role and domain views. Finally, the interaction model is built using inputs from the other views.

2.9 The Tropos software development methodology: process, models and diagrams

The Tropos methodology [9] is based on key features that are agents, goals, and plans. The phases of the methodology are early requirements, late requirements, architectural design, detailed design, and implementation. The early requirement phase identifies actors and their goals. The late requirements introduces the system-to-be as an actor that interacts

with other actors. In the architectural design more system actors are introduced and are assigned sub-goals. The detailed design defines the system actors in further details, including specifications of communication and coordination protocols. The implementation transforms the system into code compatible with the JACK platform [2].

3 Towards a Modelling Methodology for Fault-Tolerant Multi-Agent Systems

To design a fault-tolerant multi-agent system, we need to define a methodology that considers common agent concepts such as role, task, goal, etc., and also specific concepts dealing with fault-tolerance. To this end we first present the main commonalities of the different methodologies that we have reviewed in order to identify common agent concepts. Second, we present specific concepts of fault-tolerant multi-agent systems that we need to introduce to the proposed modeling methodology.

3.1 Commonalities between MAS methodologies

The MAS design methodologies presented in section 2 share common phases (analysis and design). However, no methodology has specified a development phase to implement the designed MAS. To this end, they only refer to object oriented methodologies. In addition, most of the methodologies share concepts such as role, group, interaction and environment, despite the fact that some methodologies are specialized in particular domain applications such as decision support.

A summary of the different models proposed in the reviewed methodologies is presented in what follows:

1. MAS-CommonKDAS:
   - Analysis: agent, task, coordination, knowledge, organization
   - Design: design

2. Gaia:
   - Analysis: role, interaction
   - Design: agent, services, acquaintance

3. Soda
   - Analysis: role, resource, interaction
   - Design: agent, society, environment

4. Internet Applications
   - Analysis: Gaia analysis models + social laws
   - Design: Gaia analysis models + coordination

5. AALAADIN
   - Analysis: possible roles, interactions, structure of groups
   - Design: agent organization

6. ADELF
   - Analysis: environment, class diagram, sequence diagram, activity diagram
   - Design: agent, non cooperative situation model, system architecture

7. SABPO
   - Analysis: role, interaction
   - Design: ontology, agent, detailed interaction

8. Message/UML
   - Analysis: organization, goal/task, agent/role, interaction, domain
   - Design: none

9. TROPOS
   - Analysis: actors and their goals, the system-to-be
   - Design: actor systems

We notice that there are 48 models proposed by the different methodologies we examined. The agent model and the interaction model (interaction, communication or cooperation models) appear in all the reviewed methodologies. Moreover, the role model and the organization model appear in six methodologies. These models count for 32 models out of 48. Since we aim at developing a MAS modeling methodology that is as general as can be, we will consider them in our methodology for fault-tolerant MAS. We will also consider the environment diagram in our methodology since two methodologies (SODA and ADELF) use it, and we showed its importance when designing a MAS [14] [12].

But first, let us summarize the major concepts referred to by the different methodologies, and their definitions, tha tare presented hereafter:

- Agent: An agent is a computational process that implements the autonomous, communicating functionality of an application. Typically, agents communicate using an Agent Communication Language [8].

- Task: A task refers to a set of coherent activities that are performed to achieve a goal in a given domain [5].

- Goal: A goal is a set of states of the world that an agent is committed to achieve/maintain. Therefore a goal is a situation, but not all situations are goals. A set of states of the world can be seen as a goal if there is an agent committed to achieving/maintaining this set of states [8].

- Interaction: The communication pattern performed by instances playing the roles to accomplish the task. [21]

- Collaboration: Collaboration is concerned with the interactions between agents in a multi-agent system. It is based on the relationships between the individual agents' mental structures and the system's (also seen as an agent) collective mental structure [8].

- Environment: The environment of an agent refers to all the elements that are external to the agent. One can distinguish the social environment of A (the agent that it knows) from its physical environment (the material resources that can be perceived by the agent or used by its effectors) [8].

- Organization: An organization provides a framework for activity and interaction through the definition of roles, behavioral expectations and authority relationships (e.g control) [7]. Also, from [22] we have that an organization is a collection of roles, that stand in certain relationships to one another, and that take part in systematic, institutionalized patterns of interactions with other roles.

- Role: The characteristic and expected social behavior of an agent. A role can interact with another role [8]. Another definition of a role is a set of tasks grouped semantically [13].

- Resource: a resource defines an abstract access mode, modelling the different ways in which the service it provides can be exploited by agents [8].

- Group: A group is a set of two or more agents that are related via their role assignments, where these relationships must form a connected graph within the group. Agents and Roles are associated with Groups to provide context. [16]

3.2 Modelling Methodology for Fault-Tolerant Multi-Agent Systems

This methodology review has led us to the idea that it is possible to propose a modelling methodology for Fault-Tolerant Multi-Agent Systems that groups the major development phases and concepts used by agent oriented software engineering methodologies found in the literature. This methodology is composed of two phases: analysis and design. For each phase, we will present its diagrams and how they can help to design fault-tolerant multi-agent systems.

A multi-agent system interacts with its environment, so we need a model to represent the environment; this is done in the environment model. Each agent in the system has one or more roles to play. So we need to define the different roles that will be played by the agents; this is done in the role model. Each role interacts with another role, so we have to model roles interactions; this is done in the interaction model.

So the analysis phase has at least the following models:

- An environment model that describes the environment and its evolution over time. The environment model structures the environment as a set of discrete situations described by sets of parameters. We can identify, during the design phase, particular situations that could lead agents to failure, and propose solutions to overcome the undesirable situations long before implementation, which reduces considerably the subsequent cost of system repair (maintenance). We can identify also situations that the MAS has to reach in order to achieve its objectives. These are goal situations.

- A role model that specifies the different roles that will be played by agents. A role can be seen as a set of tasks grouped semantically [13]. Each situation in the environment model could generate an agent failure. However, all identified roles must be fulfilled despite this failure, requiring a reassignment of roles between the remaining agents, which implies that the MAS organization is dynamic. Furthermore, relationships between roles do not change since they are defined according to the nature of the tasks composing the roles. Hence, the role model describes the tasks and

the necessary protocols between them; while the agent model, defined in the design phase (below), presents how roles can be dynamically assigned to agents.

- An interaction model that specifies both how roles interact with each other according to the protocols used in the interactions.

Each agent will be assigned one or more roles. So we need a model to represent agents and their roles; this is done in the agent model. An agent has to communicate with other agents. So we need to model agent communication; this is done in the communication model. Also, agents has to collaborate with each other. So we need a model to represent agent collaboration; this is done in the collaboration model.

So the design phase has at least the following models:

- An agent model that shows the agents' social roles and relationships. In the agent model, we focus on assigning and determining agents roles and agent relations. In fault tolerant MAS, we can sometimes use replicated agents to replace unavailable agents [10]; at other times, we cannot. To overcome this situation, agents can see their roles changing in order to fulfill the missing roles. We have to propose guidelines to decide how to assign unfilled roles to existing agents or how to modify roles so that the overall system still accomplishes its tasks.

- A communication model that specifies the communication paths between agents,

- A coordination model that specifies the protocols used to coordinate agents' actions. These protocols can be inferred from the interaction model.

The different proposed models consider all agent concepts presented above as showed in what follows:

- environment model: environment, goal
- role model: role, task, organization
- interaction model: interaction
- agent model: agent, group
- communication model: resource
- collaboration model: collaboration

Moreover, we may need to provide this methodology with other models that were not considered in the kernel so that it will be useful for a wide range of applications. By doing so, we hope to propose a flexible and easily adaptable methodology. We will address this issue in a forthcoming paper.

4 Conclusion

Many agent oriented software methodologies were proposed in the literature. However, none of these methodologies are fully suited for the design of fault-tolerant multi-agent systems. To overcome this problem, we presented in this paper an overview of agent oriented software engineering methodologies and identified some models that could compose the kernel of a modelling methodology for fault-tolerant MAS. This kernel is composed of two phases: analysis and design. Each phase produces a set of models as introduced above.

As future work, we wish to develop guidelines to help the knowledge engineer to produce the different models of the kernel. We also wish to address the problem of finding a formal representation of these models as one of our longer term objective is to develop a modeling environment that would encompass some automatic validation feature that would provide additional help to the system designer. For this, we will consider existing formalisms such as those proposed in AUML [15].

This work defines the basis to define an agent-oriented methodologies for fault-tolerant multi-agent systems. The different proposed models consider the different agent concepts presented above. This make this methodology genereic and appliable to a wide range of applications.

5 References

[1] Bernon, C., Gleizes, M.P., Peyruqueou, S., Picard, G. ADELF, a Methodology for Adaptive Multi-Agent Systems Engineering. Workshop Notes of the Third International Workshop Engineering Societies in the Agents World. Madrid, Spain (2002) 21--34

[2] Busetta, P., Onnquist, R., Hodgson, A., Lucas, A. JACK Intelligent Agents-Components for Intelligent Agents in Java. Technical Report TR9901, AOS (1999)

[3] Caire, G., Coulier, W., Garijo, F. J., Gomez, J., Pavon, J., Leal, F., Chainho, P., Kearney, P. E., Stark, J., Evans, R., Massonet, P. Agent Oriented Analysis Using Message/{UML}. Agent Oriented Software Engineering (AOSE) (2001) 119-135

[4] Dikenelli, O., Cenk Erdur, R. SABPO: A Standards Based and Pattern Oriented Multi-Agent Development Methodology. Workshop Notes of the Third International Workshop Engineering Societies in the Agents World. Madrid, Spain (2002) 57--70

[5] Duursma, C., Olsson, O., Ulf, S. Task Model Definition and Task Analysis Process. Technical Report KADS-II/VUB/TR/004/2.0. Esprit Project P5248, Free University Brussels and Swedish Institute of Computer Science (1994)

[6] Ferber, J., Gutknecht, O. A meta-model for the analysis and design of organizations in multi-agent systems. In Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98), Paris, France,(1998) 128-135

[7] Ferber, J., Gutknecht, O., Michel, F. From Agents to Organizations: an Organizational View of Multi-Agent Systems. Third International Conference on Autonomous Agents and Multi-Agent Systems, Sydney, Australia (2003)

[8] FIPA Methodology Glossary: www.pa.icar.cnr.it/~cossentino/FIPAmeth/glossary.htm

[9] Giunchiglia, F., Mylopoulos, J., Perini, A. The tropos software development methodology: processes, models and diagrams. Proceedings of the second international joint conference on Autonomous agents and multiagent systems, Bologna, Italy (2002) 35-36

[10] Hagg, S. A Sentinel Approach to Fault Handling in Multi-Agent Systems. Proceedings of the Second Australian Workshop on Distributed AI, in conjunction with the Fourth Pacific Rim International Conference on Artificial Intelligence (PRICAI'96). Cairns, Australia, (1996)

[11] Iglesias C. A., Garijo, M., Centeno-Gonzalez J., Velasco, J. R. Analysis and Design of Multiagent Systems Using {MAS}-Common {KADS}. Agent Theories, Architectures, and Languages (1997) 313-327

[12] Mellouli, S., Mineau, G., Moulin, B. Multi-Agent Systems Design.Workshop Notes of the Third International Workshop Engineering Societies in the Agents World. Madrid, Spain (2002) 127-138

[13] Mellouli, S., Mineau, G., Moulin, B. Towards An Agent Modelling Methodology for Fault-Tolerant Multi-Agent Systems. The Fourth International Workshop Engineering Societies in the Agents World", October, London, United Kingdom (2003)

[14] Mellouli, S., Mineau, G., Pascot, D. The integrated modeling of multi-agent systems and their environment. Proceedings of the first international joint conference on Autonomous agents and multiagent systems. Bologna, Italy (2002) 507--508

[15] Odell., J., Parunak., H., Bauer, B. Extending UML for Agents. Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence. (2000)

[16] Odell, J., Van Dyke Parunak, H., Fleischer, M. The Role of Roles. Journal of Object Technology, January-February, Vol.2, No. 1 (2003) 39-51

[17] Omicini, A. SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems. Workshop on Agent Oriented Software Engineering. (2000) 185-193

[18] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W. Object-Oriented Modeling and Design. Prentice-Hall International Editions, New Jersey (1991)

[19] Schreiber, A., Wielinga, B. J., Akkermans, J. M., Van de Velde, M. A comprehensive methodology for KBS development. Deliverable DM1.2a KADS-II/ M1/RR/UvA/70/1.1, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels.

[20] UML. Unified Modelling Language. http://www.uml.org.

[21] UML 1.5 Specification: www.omg.org/technology/documents/formal/uml.htm.

[22] Van Lamsweerde, A., Leiter, E. Handling Obstacles in Goal-Oriented Requirements Engineering. IEEE Transactions on Software Engineering, Vol. 26, No. 10, October (2000) 978-1005

[23] Wooldridge, M., Jennings, N. R., Kinny, D. The Gaia Methodology for Agent-Oriented Analysis and Design. Journal of Autonomous Agents and Multi-Agent Systems, volume 3, number 3. Kluwer Academic publishers (2000) 285--312

[24] Zambonelli, F., Jennings, N. R., Omicini, A., Wooldridge, M. Agent-Oriented Software Engineering for Internet Applications. Coordination of Internet Agents: Models, Technologies, and Applications.

Springer-Verlag: Heidelberg, Germany. A. Omicini and
F. Zambonelli and M. Klusch and R. Tolksdorf (2000) 326-346

System Administration Using Software Agents

Shahram Rahimi and Santosh Ramakrishna
Department of Computer Science, Southern Illinois University
Mail Code 4511, Carbondale, Illinois – 62901, USA.
{rahimi, srama} @cs.siu.edu

Keywords: Agents, System administration, Distributed systems**.**

Received: August 15, 2003

The increasing complexity and difficulty of system administration has been long recognized. Studies indicate that because of the difficulty and complexity, the cost of administering systems is ten times the cost of the actual hardware. Here, we present ABSA; ABSA is an agent-based solution to automated system administration. ABSA architecture is introduced to minimize the cost of administering computers in multi platform networks and to provide a simple, consistent, expandable and integrated system administration tool. ABSA system supports important system administration features such as domain-wise administration, automated error handling and default system configuration besides others.

1 Introduction

Networks maintained by many sites today contain tens to hundreds of computers. Managing such a sizeable collection of computers and their software is a challenging task, generally referred to as system administration. Majority of the tasks performed by a system administrator on a day to day basis include ensuring all hardware and software is in working order, managing user accounts, dealing with the security threats, backups, software upgrades, maintenance, recovery from system failure and ensuring an adequate supply of resources such as swap and disk space. Performing all these tasks manually can prove to be very difficult, especially when dealing with a sizable collection of computers. Majority of the day to day activities performed by system administrators are procedural and recurring and hence a burden to the system administrator [1]. This complexity and difficulty of system administration has been long recognized. Studies indicate that because of complexity, cost per year of administering systems is much higher than the cost of the actual hardware itself [2]. While system administration is challenging and burdensome, most of the tasks performed by an administrator can be automated to great extent. Moreover there is a limit on the number of systems that can be maintained by an administrator, which highlights the need for a scalable approach.

In this paper, we present an agent-based architecture to facilitate and automate the system administration tasks. Distinctiveness of agents such as autonomous nature, intelligence, perseverance, adaptability, and of course mobility are most appropriate for their use in our architecture. The mobile nature of agents allows keeping minimum essential environment on the remote host that is just enough to allow execution of agents on it. This avoids the concentration of the operations in a single computer; instead, it uses the computing power of other computers by distributing the tasks. Moreover, using java agents in ABSA provides the system with platform independency which further distinguishes ABSA from other tools available in the market.

The remaining parts of the paper are organized as follows. First, a brief background on different system administration approaches and software agents is given in section 2. Then the general architecture of the system is presented in section 3. In section 4, we describe the implementation and the tools used. Finally, a brief summary concludes the paper.

2 Background

In this section, we briefly review the current centralized system administration approach and discuss some of the existing tools that aid system administrators.

2.1 Centralized System Administration

Recently, there has been considerable amount of research to replace the traditional ad hoc system administration by client/server based applications, which aim to centralize the process. These centralized applications use mainly two protocols, the Simple Network Management Protocol (SNMP) and the Common Management Information Protocol (CMIP). Both protocols follow a client/server approach with managers invoking operations on management programs. They also provide mechanisms for reporting events by management programs. However, there are fundamental differences between these two protocols. CMIP offers a much richer set of protocol operations both on manager and on management. However SNMP is a simpler tool for and is more popular in the market.

SNMP, CMIP, and related approaches to network and system administration are centralized paradigms based

on the client/server architecture. These solutions require gathering all management functionality in a central manager which causes complexity and lower performance. Moreover, they do not address heterogeneity of the platforms. Scalability is another disadvantage of centralized approach, which loses performance to the size of the network.

2.2 Analysis of existing System Administration Tools

Automated administration of systems is becoming increasingly important due to the associated costs. Some work has been done in this regard to either partially automate the tasks or develop tools to aid administrators.

"Software Update via Mobile Agent Based Programming" [3] is one such approach for automated updating of software on the systems. This model has some limitations such as platform dependency. Moreover, the software has to be maintained on the server, which causes centralization of considerable amount of the tasks and hence a bottleneck. As a second example, we can name "The Igor System Administration Tool" [4] a tool for performing administration tasks simultaneously on numerous hosts. Although it eases the task of system administration, it does not deal with automation of system administration and it focuses on UNIX systems only.

"Central System Administration in a Heterogeneous UNIX Environment: GeNUAdmin" [5] is another example. In this tool, configuration profiles for clients are maintained on the central server and clients are configured based on their profile on the server. Administrators have to modify the configuration files on the server to manage the clients. The modifications are automatically transferred to the client systems. Its disadvantages are that it may cause inconsistency among configuration files on the server and the actual client configuration and also it is for UNIX systems only. Our last example is "WEBMIN: A Web-Based System Administration Tool for UNIX" [6], which is a web based tool for configuring UNIX systems. This one does not support platform independency either.

All the above tools are based on client-server architecture. This makes them less scalable since all the administrative tasks are done on a single computer.

2.3 Software Agent Technology

A definition of "software agent" that many agent researchers might find acceptable is: a software entity which functions continuously and autonomously in a particular environment, often inhabited by other agents and processes [7]. The requirement for continuity and autonomy derives from our desire that an agent be able to carry out activities in a flexible and intelligent manner that is responsive to changes in the environment without requiring constant human guidance or intervention. In general, software agents are differentiated from other applications by their added dimensions of mobility, autonomy, and the ability to interact independent of their user's presence.

There are two types of agents, namely stationary agents and mobile agents. Stationary agents are permanently attached to a place (node), while mobile agent can move from one place to another. An agent is said to be strongly mobile if its entire code and execution state move with it.

In our architecture, we use stationary agents for management purposes and mobile agents to distribute the system administration tasks among the computers in the network. Agent technology provides a fresh scalable approach to system administration, which avoids the difficulties of the traditional client-server approach.

3 Agent-Based System Administration

3.1 General System Architecture

In this section, we present the architecture and the behaviour of ABSA. We divide the computers present in a network into two categories, namely the central manager node, from which we manage other nodes in the network, and the client nodes that are managed by the central manager node. The central manager node is responsible for receiving the administration requests, analysing the requests and dispatching necessary agents to appropriate client machines to carry out the request(s) and report the status.

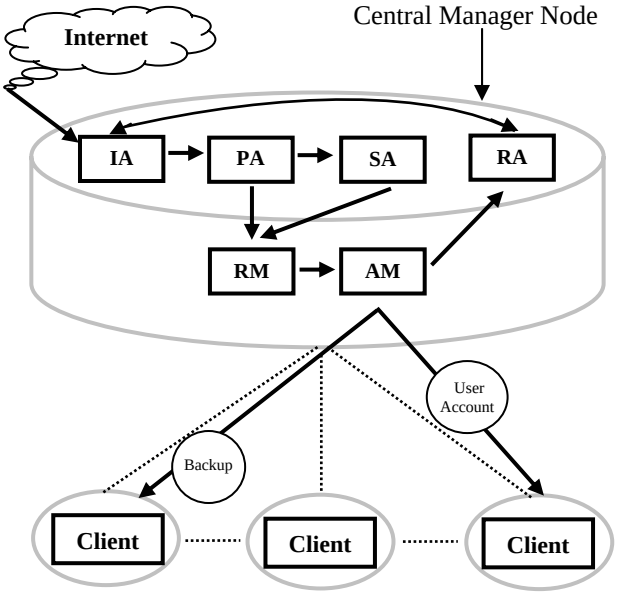


Figure 1 System Architecture

Within this overall architecture, there exist multiple agent classes, both stationary and mobile, and including

both intelligent, and less intelligent software agents. Central manager node has different stationary agents within itself to perform the necessary tasks. The only mobile agents in this architecture are the Action Agents which migrate to the client nodes to perform the requested tasks. We refer the reader to Figure 1 for the following discussion of the architecture in a network of heterogeneous systems.

Figure 1 illustrates the general system architecture. Before progressing to describe how the system operates, we list different agents that at this moment are used in the system together with a brief description of each.

Internet Agent (IA): It receives administration requests and also requests for the status of submitted tasks via internet and is actually the server side for web-based GUI. The IA is a stationary agent on the central manager node. For each submitted task IA generates unique ID that could be used at a later time to find the status of the task. IA sends the submitted administration requests to the Processing Agent and status related requests to Report Agent.

Processing Agent (PA): Receives requests from IA. PA deciphers if the task is one time task or a scheduled task. If the task is scheduled one, it is sent to the Scheduler Agent, else it is sent to the Request Manager using appropriate protocols. It is also a stationary agent on central manager node.

Scheduler Agent (SA): It is a stationary agent on the central manager node. Responsible for generating requests to the Request Manager for scheduled tasks and managing the tree data structure used to keep information about the scheduled tasks.

Request Manager (RM): Maintains the request queue on a priority basis. It could receive requests from PA or SA depending on the type of the task. It is stationary agent on the central manager node.

Agent Manager (AM): Responsible for generating mobile agents in the system to carryout the requested tasks. It receives a task from the RM and generates an appropriate action agent to perform the task. It then moves the action agent to the client on which the task has to be carried out. AM is again a stationary agent on the central manager node.

Report Agent (RA): It is a stationary agent on the central manager node. RA is responsible for maintaining status of the tasks being managed by the AM. RA also maintains the log file for all the submitted tasks and their current status. RA processes the log file and provides results for status related queries by IA.

Action Agent (AA): These are mobile agents generated by the AM to perform the requested task. AA is a broad term given to a set of task-oriented agents. There are different action agents for different tasks. AA migrates to the client machine, performs the requested task and informs AM about the status.

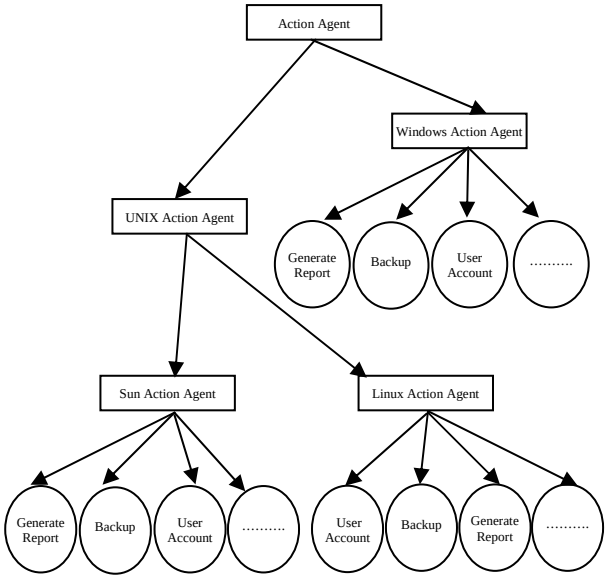


Figure 2 Hierarchies of Action Agent Classes

As stated earlier, there are different action agents for different tasks. Figure 2 shows the Hierarchies of Action Agent classes. At the highest level we have Action Agents that could perform tasks independent of the operating systems. As we traverse hierarchy of action agent classes downwards, we have action agents which are very specific to operating system. The user account agent class under sun action agents is specific to sun operating system while the user account agent class under windows action agents class is specific to windows operating system. Action agents are sent out to the target systems based on the type of the operating system installed on it.

3.2 System Behaviour

At this stage of the system implementation, we have only focused on automation of major routine tasks such as managing user accounts, backup, upgrading application software, applying patches, antivirus updates and checking printer status. This section describes a typical scenario that utilizes the above named agents for system administration.

As it was mentioned earlier, the Processing Agent receives requests from the Internet Agent. Since these requests may be simultaneous, the PA maintains a FIFO queue for the inputs. It decodes the task requests and sends them to the Scheduler Agent or the Request Manager based on the type of the task, using appropriate protocols. If the task needs to be scheduled, it would be send to SA; otherwise, one time tasks are sent to RA.

The IA provides a web-based GUI and is used for submitting tasks as well as viewing their status. Upon submission of a task request, the user is given a unique

task ID. The task ID is generated based on the current time (including month and year, in order to generate a unique ID), and the user can later use this ID to obtain the status of the submitted task. The IA gets the status of the task from the Report Agent.

The Scheduler Agent preserves a two level tree structure in which the first level contains the hostnames of the computers in the network and the second level includes the scheduled tasks for each computer. Each node in the first level of the tree, in addition to the hostname, holds the next immediate scheduled task. The next level of the tree maintains the list of the scheduled tasks to be performed on each host. This is illustrated in Fig. 3. Whenever a scheduled task is picked for operation or a new task is added to the tree, the SA searches the second level to find the next immediate task for each node and place it at the first level by the hostname. This is done in order to reduce the search time.

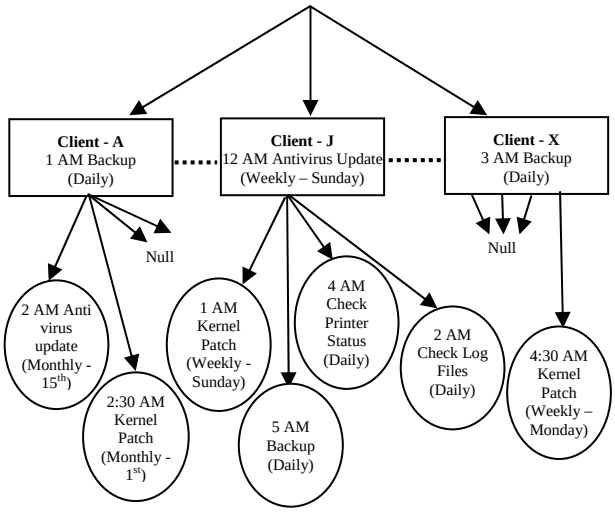


Figure 3 Scheduler Agent Data Structure

The Request Manager receives the task requests from PA (one time tasks) or SA (scheduled tasks). It maintains a priority queue of the requests. The priorities are assigned based on the origin and the significance of the requests. If the origin of a request is a regular user, its priority is less than that of a request from the administrator. In addition, the priority of an "antivirus definitions updating" task is higher than the priority of a "create user account" request. The system has a default priority setting; however, the administrator can change these priorities.

The Agent Manager has a threshold on the number of Action Agents it can maintain at a time. When the number of AAs in the system is less than the threshold, the AM accepts new tasks from the RM and creates appropriate AAs to be dispatched to the corresponding client computers. After creating an AA, the AM sends its task request to the RA which assigns the "in progress" status to the task. Upon completion (or failure), AA reports the status to the AM (either "completed" or "error" with a code number). AA will be suspended after completion of its task. AM then updates the status of the task with RA.

As it was mentioned, there is a different AA for each of the tasks. For instance, for creating a user account we have User Account AA, for updating antivirus definitions we have Antivirus AA and so on. AAs are the only mobile agents in the system and most of them have some level of intelligence.

An example of one time task such as create user account will go through the following sequence of agents in the order specified: IA followed by PA, RM, AM, AA and RA. An example of scheduled task such as backup will also go through the same sequence of agents except that SA is in between PA and RM since it considered a scheduled task.

Domain-wise Administration

One of the key features of ABSA is its support for domain-wise system administration. In case of large computer networks, computers are logically grouped together to form domains. The domains themselves can be grouped together to form a higher level domain. This logical grouping of systems makes domain administration tasks possible. For instance the administrator could create a user account for a particular domain, which can be used on all the computers in that domain.

We now discuss how domain-wise administration is achieved in ABSA. We maintain the domain information of the network in a tree structure. As shown in Fig. 4 each node in the tree contains a domain name, a domain ID and the list of all the users allowed administering the systems in that domain.

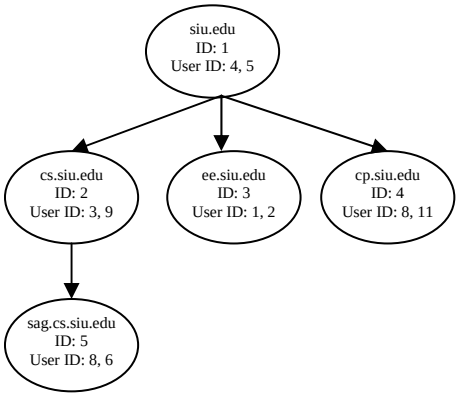


Figure 4 Domain Information Data Structure

From figure 4, users with ID 4 and 5 have same administrative rights for the domain siu.edu and all the domains under it. User with ID 3 can administer computers in cs.siu.edu domain and the domains below it; user with ID 6 can administer computers in domain sag.cs.siu.edu. Users with rights for a domain cannot administer systems in the higher levels; viz. user with ID

6 cannot administer computers in cs.siu.edu or siu.edu domain. ABSA gives the granularity of granting users to perform only a subset of administrative tasks. The following paragraph discusses how this granularity is achieved.

When a user submits a task, user enters the domain name he/she is willing to administer; the system obtains this domain name and his/her user ID. We then perform a traverse on the domain information data structure (shown in Fig. 4), find the domain's node, and check if the user is allowed to administer the systems in the domain, if the user is not authorized to perform the requested task in that domain, his or her request will be denied. Furthermore to set up user permissions, we maintain a profile for each user. Each entry in this profile contains a domain ID followed by a 32 bit number, each bit of this number determines whether a user is allowed to perform a particular task or not depending on whether the bit is set or reset. If the user is allowed to perform a particular administration task on a domain (after checking the domain information data structure), his or her profile is checked for that particular domain ID and whether he or she is authorized to carry out the particular task that he is submitting. Depending on the result of the profile check, his/her request may be accepted or denied. All the above checks are performed by Internet Agent before accepting the user request.

Default System Configuration
ABSA supports configuration of computers from scratch (computers installed with an operating system and an agency to support agents). Administrator maintains a default configuration profile for every domain, which specifies the steps to be carried out and the agents to be sent out. This default configuration profile can be used to configure computers from scratch.

Automated Error Handling
ABSA can read the system logs generated by Windows and UNIX operating systems and take appropriate actions in response. The system logs generated by operating systems are in response to events such as information, warning and error. At this stage of the system implementation, we are handling only the error events. As discussed earlier on every client we have an agency operating, Monitor Agent in the agency reads these system logs at regular intervals of time which is a configurable parameter.

We now discuss how the automated error handling works. The Monitor Agent checks to see if there are any new errors in the log file from the previously read time. If any new errors are found the Monitor Agent communicates with the Internet Agent about the error. The central manager then performs a check in its knowledge base to see if it has a similar error in its knowledge base and the solution for the error. If no match is found for the error in the central manager's knowledge base the error is reported to the administrator. If a match is found appropriate agent is sent from the Agent Manager to fix the error. The whole picture of automated error handling can be viewed as the Monitor Agent on the client submitting a task request and the central manager node processing the task request.

Autonomous Software Management
Another important feature that is supported by ABSA, which is at its early stages of implementation, is Autonomous Software Management. Autonomous Software Management allows application software to be deployed by the users independently in a controlled form as specified by the administrator. Administrators need only to prepare the software once for the system and then leave the entire deployment and maintenance to the system itself. This greatly reduces the workload of administrators and also improves the process of software deployment significantly.

Here is how Autonomous Software Management is being implemented in ABSA. At the central manager node we have Software Manager Agent (not shown in Figure 1), which is a central control application for software management, linked to an SQL database. Administrator has to pre-configure each software by packaging it into a JAR file [11] which contains a Manifest file [12]. Using this format allows the package to be digitally signed for security and at the same time allows the package to be compressed. The SQL database is used to consolidate information pertaining to each package, this information is called an installation profile. Each of these profiles contains a variable and a fixed component, where variable component can be modified by users after application has been installed while the fixed component cannot. Users may use a web-based interface to look up a particular package available for installation on the server.

At the client end, Monitor Agent (discussed earlier), collaborates with the Software Manager Agent to automate the process of software deployment. Monitor Agent is controlled by Software Manager Agent to perform tasks on the computer in which it resides. This agent monitors applications to ensure that their installation profile is properly adhered. It also monitors usage statistics such as how often the application is used. Moreover adjustments made by the user to the variable component of installation profile are also noted, so that the default installation profile, stored on the server, is updated.

In order to facilitate the operation of the ABSA architecture across multiple operating systems, the choice of implementation tools are vital.

4 Implementation Tools

The system is being implemented in Java and over Grasshopper agent environment, while the knowledge bases of the intelligent agents are being written in Java Expert System Shell (JESS). The choice of Grasshopper

platform and JESS were based on a comparative study of existing tools and environments [8]. Version 0.2 of the system is actually functioning and is being tested at this time.

Grasshopper is implemented completely in Java and is designed in conformance with the Object Management Group's Mobile Agent System Interoperability Facility (MASIF). The platform can be enhanced with an add-on, which is compliant with the specification of the Foundation for Intelligent Physical Agents (FIPA) [9].

JESS is a rule engine and scripting environment written entirely in Java. Jess is Java implementation of CLIPS expert system shell and is a scripting environment, from which objects can be created and methods can be called without compiling any Java code [10]. Java provides APIs for network communications, implements threads, remote procedure calls, web request processing, and also gives the system the advantage of platform independence. Therefore ABSA is capable to manage networks of different operating system platforms.

Now, we further extend this discussion to important data structures followed by some implementation details for each of the agents in the system.

One of the important data structure used in the architecture is the synchronized circular shared buffer. This buffer is used by all the stationary agents in the system to communicate with one another. Since the buffers are shared between concurrently running agents, only one agent should be allowed to access the buffer in order to maintain the buffer consistency. Java provides APIs to synchronize access of objects, which allows only one thread to access an object at a time. We use this synchronization and create synchronized circular shared buffer objects for communication.

Another important data structure is the tree structure used by Scheduler Agent, which was discussed earlier. We now extend the discussion to implementation details of the agents in the system.

Internet Agent: It is a Java Servlet which responds to web requests. IA communicates with PA and RA using Datagram Sockets. Using sockets for communication provides us with the advantage of having IA either on the central manager node or on a different web server and still be able to communicate with PA and RA. For administration related requests, IA first verifies if the necessary parameters to carry out the task are correct and consistent, then it concatenates the received parameters in a particular sequence and passes it on to PA. For status related requests, IA passes the received status related query to RA and displays the output generated by RA to the user.

Moreover multiple task requests can be batched together in a file. IA can accept batch files and pass on a request for each of the tasks in the batch file to PA. Batching is very convenient especially when a task has to be performed on multiple hosts, such as fixing bugs, installing patches, holiday shutdown etc.

Processing Agent: Receives requests from IA using Datagram Sockets. It decodes the task to be performed and then passes the task request to SA or RM depending on the task type using synchronized circular shared buffer object.

Scheduler Agent: It reads from the shared buffer object of PA and writes it into the tree data structure (discussed earlier). SA processes the tree in such a way that the tree always holds the next task to be performed at the first level. At the scheduled time, SA writes the request to the shared buffer object of RM.

Request Manager: It reads from the shared buffer objects of PA and SA, and maintains a priority queue. It shares this priority queue with AM. AM reads the topmost request from this queue.

Agent Manager: It invokes an appropriate agent class for the task and migrates the agent to the client using Grasshopper Agent Platform. It also writes the status of the tasks to shared buffer object of RA.

Report Agent: It reads from the shared buffer object of AM and updates the log file for the task status. RA can search the log file for a task ID, tasks on a particular host and tasks submitted on a particular day.

Action Agents: They are mobile agent classes. Each AA is specific to the task and to the operating platform on which the task has to be performed.

Figure 5 illustrates UML sequence diagram, depicting the flow of control between agents. This diagram describes the timing sequence of method calls between different classes. The flow of control is initiated by user request to IA. The arrows in the sequence diagram correspond to the method calls.

All the stationary agents shown in the UML diagram (Fig. 5) are java threads running in parallel. These agents communicate with each other using either datagram sockets or circular shared buffers as discussed earlier. The UML sequence diagram also depicts some of the important methods used. Flow control starts with a web request from user to IA either for performing a task or to know the status of a submitted task. The *doPost* method of IA handles these user requests and the *getTaskID* method generates a unique task ID for each task request. The *send* method of IA transfers the user request either to PA or RA depending on the type of the user request. After sending the request to PA or RA, IA waits to receive new requests.

The *send* method of IA corresponds to the *receive* method in PA which receives the task request.

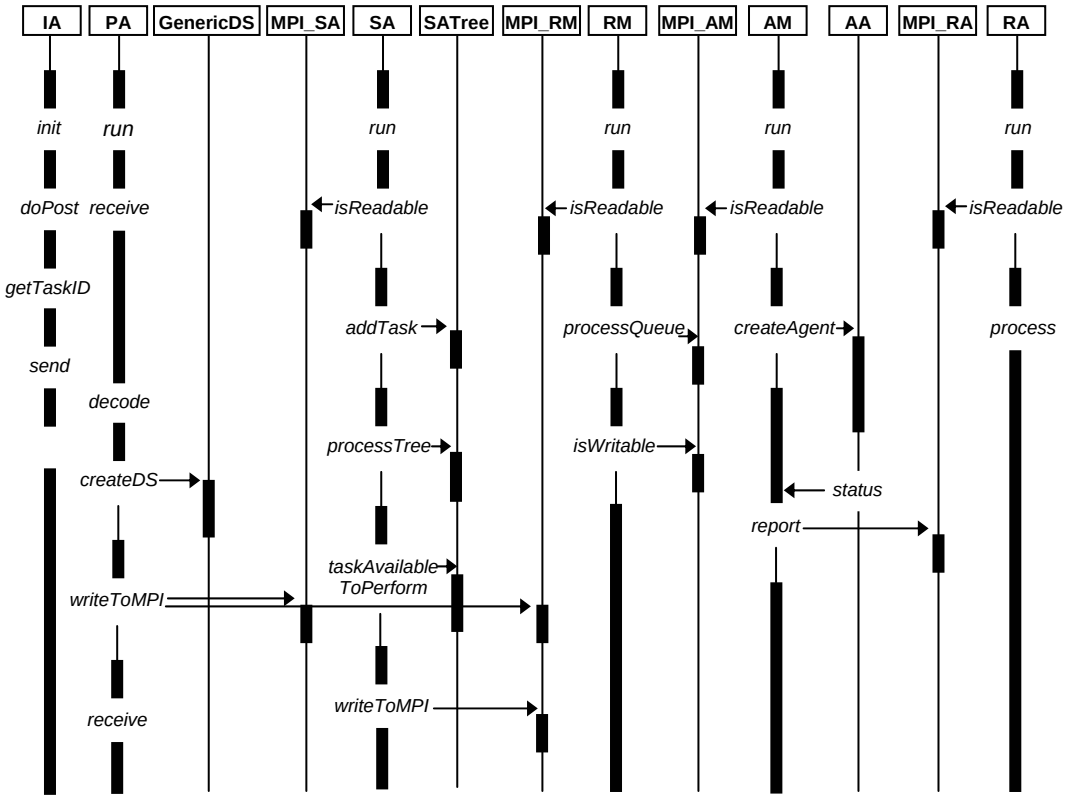


Figure 5 UML Sequence Diagram

Upon receiving the request, the *decode* method of PA determines the type of the task and calls appropriate method of the *GenericDS* class. The *writeToMPI* method of PA writes this request to shared buffer of SA or RM depending on the task (scheduled or unscheduled).

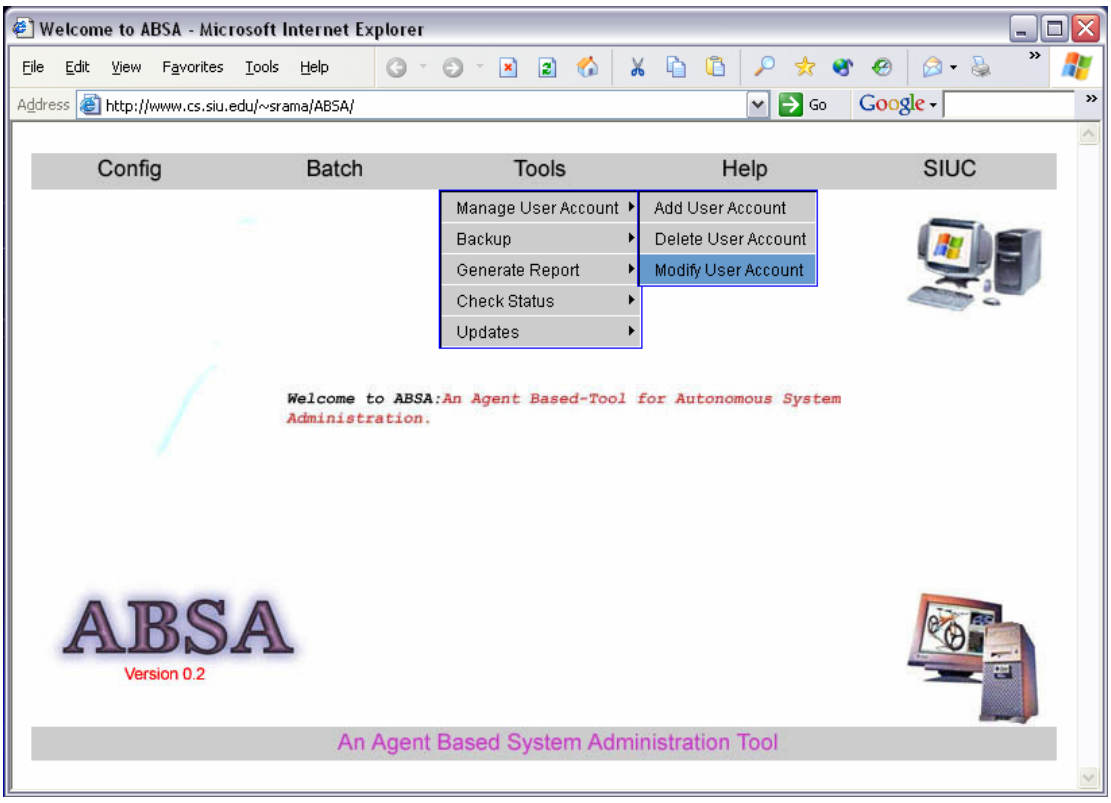


Figure 6 ABSA: Main Interface

Let us assume that the task is a scheduled one such as performing a backup. The SA reads the task request from the top of the buffer and adds it to *SATree* using *addTask* method. The *SATree* is processed by *processTree* method of SA to arrange the tasks in such a way that the next task to be performed on the host is at the first level of the tree as discussed earlier.

The *taskAvailableToPerform* method of SA periodically checks the *SATree* to find if any tasks are available to perform; if available the requests are written to shared buffer of RM.

RM reads the requests from the buffer shared with SA and PA and uses its *processQueue* method to rearrange the tasks in the queue on a priority basis. The priority queue maintained by RM is the shared buffer to AM. The *isWriteable* method of RM writes the task request to this priority queue.

The AM reads the priority queue maintained by RM using *isReadable* method. This method always reads the topmost task in the priority queue. For each task read, AM creates the appropriate AA and migrates it to the client to perform the task. AA upon completion of the task, reports the status to AM. AM writes the status of the performed task to the shared buffer of RA using its *report* method. RA reads the shared buffer using *isReadable* method and writes the status of the task to the log file, maintained by itself using the *process* method.
The *process* method also sorts the log file based on the task ID.

Figure 6 shows the web-based GUI of ABSA. System Administrators can log in from anywhere in the world and use the system. Upon choosing a task, the user gets an interface with parameters specific to that particular task, Figure 7 shows the interface for create user account.

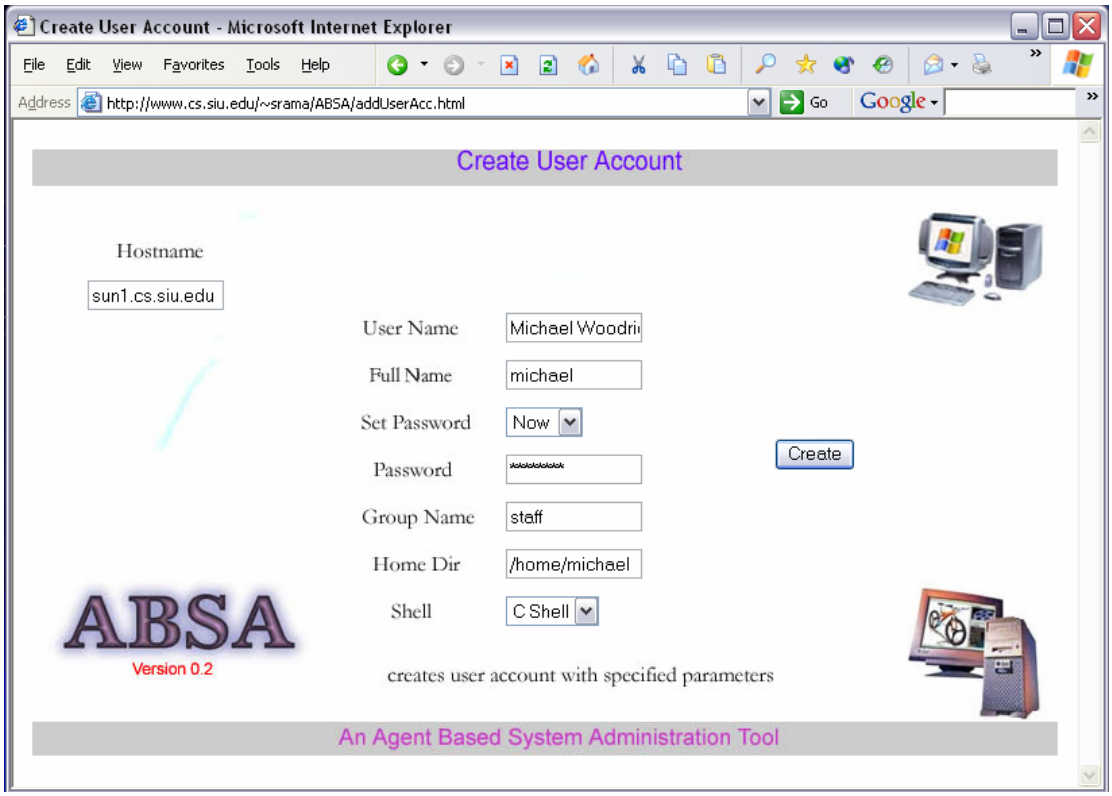


Figure 7 ABSA: Create User Account Interface

5 Conclusion

This paper presents ABSA, a new tool for automation of system administration based on a novel agent-based architecture. System administration by itself is a challenging area; besides, the added complexity of working with different platforms in a heterogeneous environment is immense. ABSA v0.2 was preliminarily evaluated against few current centralized approaches for automation of system administration tasks and the results were promising. The performance tests were based on the following criteria: expandability, extent of automation, error rate of the overall architecture, overall security of the architecture, multiplexing (distributing) of tasks, and ease of use.

The performance of ABSA v0.2 architecture was better than other approaches in many of the criteria's used for the evaluation. ABSA approach was highly scalable compared to other approaches. Upon increasing the number of computers in the network, the performance of ABSA was relatively steady, while the performance of other centralized administration approaches fell due to increased load on the central server. ABSA system does not maintain client configuration files on the central

manager node as compared to other approaches, thus avoids any possibility of inconsistency between the client and the central manager. The user interface of ABSA is very friendly and is independent of the operating platform, which allows administrators familiar with only one operating platform to administer systems of other platforms as well.

ABSA has few disadvantages such as security of the agents in the system. One more disadvantage of ABSA is that it needs to keep up with new releases of operating systems. Moreover, since ABSA supports multiple operating systems, maintenance may be needed more frequent compare to other systems. Although ABSA has some limitations, the advantages outnumber these disadvantages.

References

[1] Miller and Donnini, “Relieving the burden of system administration through support automation” Proceedings-of-the-Fourteenth-Systems-Administration-Conference-LISA-XIV. 2000: 167-80

[2] Gartner Group. A white paper on Gartner group’s next generation total cost of ownership methodology, 1997.

[3] L. Bettini, R. De Nicola, M. Loreti, “Software Update via Mobile Agent Based Programming,” Publication: 2002 ACM 1-58113-445-2/02/03

[4] “The Igor System Administration Tool”, Tenth USENIX System Administration Conference Chicago, IL, USA, Sept. 29-Oct 4, 1996.

[5] “Central System Administration in Heterogeneous UNIX Environment: GeNU Admin” LISA, pp. 1-8, September 19-23, 1994

[6] “WEBMIN: A Web-Based System Administration Tool for UNIX” USENIX Annual Technical Conference, San Diego, California USA, June 18-23, 2000.

[7] Y. Shoham, “An overview of agent-oriented programming”, Software Agents, ed J. M. Bradshaw. Menlo Park, Calif.: AAAI Press. 1997.

[8] S. Rahimi, R. Angryk, J. Bjursell, M. Paprzycki, D. Ali, M. Cobb and K. Kolodziei, “Comparison of Mobile Agent Frameworks for Distributed Geospatial Data Integration,” Proceeding of the 4th Agile Conference on Geographic Information Science, Brno, Czech, pp. 643-655, 2001.

[9] Grasshopper Programmer’s Guide, URL: http://www.grasshopper.de.

[10] Jess- the Java Expert System Shell, URL: http://herzberg.ca.sandia.gov/jess.

[11] Using JAR Files: The Basics. http://www.java.sun.com/docs/books/tutorial/jar/basics/index.html.

[12] Understanding the Manifest, http://www.java.sun.com/docs/books/tutorial/jar/basics/manifest.html.

Collaborative Translation with Mobile Agents

Eric Sanchis, Jean-Louis Selves and Zhao Yang Pan
Laboratoire Gestion et Cognition . IUT Ponsan - Université Paul Sabatier.
115, route de Narbonne. 31077 - Toulouse - Cedex. France
Email: sanchis@iut-rodez.fr, selves@meph.iut-tlse3.fr, zhaoyang.pan@libertysurf.fr

Keywords: mobile agents, actual mobility, peer to peer application, intermediate design object.

Received: August 15, 2003

In many intellectual or industrial fields, it is frequent that groups of actors have to collaborate in order to look for and to find solutions to new or old problems. The geographical distribution of the users imposes the use of a common software platform as transparent as possible. Having stated the hypothesis that the emergence of new solutions could arise by the co-location of potential solutions, we propose a lightweight peer to peer architecture based on mobile agents which implement a model of mobility called actual mobility.

1 Introduction

The information sharing became with Internet an essential practice of numerous playful [22], [14] and professional activities such as collaborative working and Concurrent Engineering [3], [21], [24].

Many experiments in industrial and research fields showed that collaborative work by way of Internet allows:

- a better coordination between the two operating modes of the users: individual work and collective work
- a faster and broader integration of human, material and informational resources coming from various horizons
- it makes easier the communications and the management of information and documents. That is possible when the systems used by the various users are compatible.

However, without calling into question the beneficial effects brought by the computer networks on the practice of collaborative work, it is easy to notice that computer means (Internet included) do not solve all the problems. Indeed, the combination of several factors condition the collaborative activity of a group of people such as the domain area, the size and mode of functioning of the group and the tools used.

For instance, the domain of application imposes more or less strong constraints on the confidentiality of the documents exchanged as well as their lifespan. In Concurrent Engineering the confidentiality must be strong and the lifespan of the documents created is generally important. In the educational field, the application that we will describe hereafter does not require a strong confidentiality and the documents created have a rather short life.

The size and the mode of functioning of the group or groups of users have also a very important influence, in particular on the characteristics of the communication protocols used. Multimedia file sharing applications or computer resources sharing as SETI@home [1] make a very significant number of users interact compared to an educational application where only a class of a few dozens of students are concerned.

Certain applications favour simultaneous interactions between the users, the others encourage asynchronous interactions.

Finally, the competences of the users and the results expected from their collaboration will condition the choice of one or several tools.

We will illustrate the consideration of the whole of these aspects in an educational environment with a software infrastructure allowing a group of students to practise a foreign language by means of the collective translation of texts.

The next section of this paper presents some general solutions adopted in the educational world as well as the global functioning of our application of emergent translation. Section 3 gives an overview of the main concepts introduced by the application model and describes its architecture. Design and implementation choices relative to the mobility of the agents and their architecture are discussed in section 4. The next section presents a scenario which will illustrate the execution of the whole. Finally, the last section presents the application software.

2 A Collaborative Application in an Educational Context

2.1 Related works

In the educational field, new collaborative tools appeared, associating mainly web technologies and peer-to-peer (P2P) architectures. This association presents various forms: the use of a P2P centered generic architecture [5], interconnection of collaborative tools

with a general purpose like Groove [9] or the packaging of suitably selected communication tools [4].

The purpose of the Edutella project [18] is to specify and implement a set of generic services based on the RDF metadata concept and the JXTA P2P framework. This infrastructure [8] is composed of a set of XML based communication protocols and provides a set of functionalities useful for the implementation of P2P applications. An Edutella application, whatever field it concerns (educational or other), uses three layers of software: JXTA architecture, Edutella services and the software specific to the application.

Applications using non-specialized collaborative tools or an assembly of communication tools aim first at increasing the interactions between the participants, who communicate in an instantaneous way and have a shared workspace. Users are geographically scattered and interact in a synchronous or asynchronous way. The main functionalities integrated in this type of collaborative tools are chat function, contact list management, calendar, whiteboard and file sharing. This combination of communication tools offering a maximum interactivity is supposed to guarantee the largest efficiency.

2.2 A Collaborative Translation Application

The design of our collaborative application is based on different premises. Indeed, we think that in certain applications, the interactivity between the users must be perfectly controlled. This aspect is particularly important when the main objective of the application is to allow a student to lead a personal work suitably while benefiting from the work of the other participants. It requires certain constraints on the tools to be used. In particular, collaborative tools integrating natively the chat function or giving to all the users a shared space as whiteboard do not seem adapted to the educational objective previously expressed.

Compared to the tools introduced above, our application called CTA (Collaborative Translation Application) has the following characteristics:

- it provides the means necessary to the control of the services of communication between the users
- its software infrastructure is light.

This application works as follows: a language teacher publishes a text to be translated intended for an *open class* of students. The expression *open class* means that 1) students can be geographically scattered, 2) anonymity covers the identity of the students, 3) users outside the class of the teacher can participate in the collective activity of translation.

Students can download the text to be translated and can publish a proposition of translation. Then, each participant (teacher or student) has the possibility of reaching the translations suggested by the others. Indeed, we make the assumption that 1) the development of a translation may be considered as a process of search of a solution through interactions among several actors, 2) the emergence of a better translation produced by a student can occur by the presence on its site of a set of co-located translations produced by other participants.
Two main services are implemented by the CTA software infrastructure:

- the tracing of the original document to be translated and recovery by the teacher of the translations suggested by the students
- the recovery by a student of the translations suggested by the other students.

On the other hand, to favour the integration of new tools within a group of users, at least two conditions seem necessary 1) the software infrastructure must be light, 2) it must disturb as little as possible the practices and working methods of each one.

To satisfy the first condition, an application must offer few but well defined services i.e. the usefulness of which must be obvious. To satisfy the second, the functioning of the application must be asynchronous: that means that the user asks for the execution of a service but that it does not remain blocked until its termination.

That led us to the following software architecture: each user (teacher or student) reaches the application via a content management system. All these autonomous systems are interconnected by means of a specialized communication layer.

The next section introduces the concept of Intermediate Design Object and describes the application architecture.

3 Application Model

3.1 Intermediate Design Objects

Introduced in Concurrent Engineering by Jeantet and Boujut [10], the *Intermediate Design Objects* (IDO) are all the concrete and abstract objects which are produced or used during the action of design and which connect tools, procedures and actors. The great applicability of the IDO concept allowed us to define a minimal model of an IDO but perfectly operational within the framework of a collaborative educational activity.

IDO structure - The model of IDO we have chosen contains two parts: a main element, called *document* which is the visible part to the users of the IDO, and a composite part called *history*, which is partially or totally masked to users according to the application.

For the user, the document corresponds to the main aspect of the IDO. Indeed, it is this element which triggers the actions of the users (consultation, modification, additional contributions). In CTA, the document is composed of the text to be translated supplied by the teacher.

The second element synthesizes all the enrichments which were brought to the first component by the members of the project, the history of the possible copies and its movements. In other words, it summarizes in a global and synthetic way the state of evolution of the

IDO at a given moment. In CTA, a history is composed of two parts: 1) a translation possibly suggested by a student, 2) the log file memorizing all the operations carried out.

Services - In CTA, four services were defined on the IDO: the *get* service, *change* service, *propagation* service and *retropropagation* service.

Get service allows a student to download the text to be translated. This service uses a simple mechanism of file transfer. The log file of the downloaded document is modified by the *get* service.

Change service is a local service authorizing the creation or the modification of a translation. This service modifies the two parts of the history of an IDO.

The *propagation* service is usable only by the teacher. It enables him to create the tree of diffusion of the text to be translated and to retrieve the propositions of translation of the students. This service provides a vertical / hierarchical treatment of the IDO.

The nature of the CTA architecture also authorizes a horizontal / peer to peer treatment of the IDO. The r*etropropagation* service makes it possible to each student to analyse the history of its local IDO, to reach the IDO of the other students and to acquire a copy of their translation.

Propagation and *retropropagation* services are implemented using mobile software agents.

3.2 CTA Framework

The CTA includes two parts: the *user interface* and the *mobile agent layer* (Figure 1).

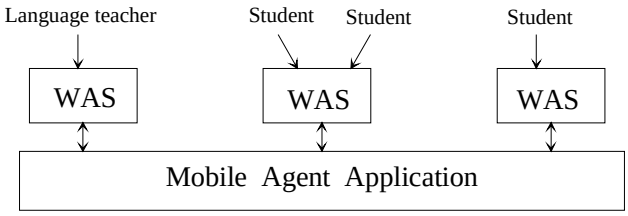


Figure 1 - CTA architecture.

We use a web application server (WAS) as *user interface*. A WAS is an important part of more powerful systems called *web content management systems* (WCMS). A WCMS provides all the tools necessary to

- the creation, edition, modification and storage of contents intended for the publication on the web,
- the management of the permissions of the users and
- the management of the interface with the user.

In CTA, the *user interface* allows the language teacher and students 1) to publish original texts and translations, 2) to transparently interact with the *mobile agent layer* when *propagation* and *retropropagation* services are used.

The Mobile Agent Application is a communication layer which interconnects web applications servers.

To structure this layer, three models are usable [7]:

- *remote code invocation* (often called *client/server* model): the local code calls remote code which executes the requested service and returns the results to the local code,
- *remote code evaluation*: the local code sends the code to be executed on the remote site which evaluates the received code, returns the results to the local code and suppresses the received code,
- *mobile agents*: the local code sends an autonomous code to the remote site which executes it. During its execution the received code has the possibility to transfer itself on another site.

We chose the third model to build the CTA communication layer. Section 4.3 will precise the different reasons of this choice.

To implement *propagation* and *retropropagation* services the *mobile agent layer* manages and executes two classes of active entities 1) the mobile agents launched by users, 2) the mobile agent servers which receive incoming mobile agents and manage their local execution.

The next section will describe two main characteristics of this layer: a *lightweight peer to peer architecture* and the *actual mobility* model.

4 Mobile Agent Application

The designer of distributed applications has two main models of architecture: *the client / server model* and the *P2P model*.

In the client / server model the client and the server have completely different functionalities. Let us take two examples: the *telnet* client program makes it possible for a user to be connected to a remote machine when it runs a *telnetd* server process. In the client / server model the server process is called daemon because this program must run continuously. In a web application, a browser is a client program which interacts with a web server.

In the P2P model, two communicating programs implement the same features. In fact, there is no client program and server program anymore but a single program offering a client function and a server function. To be more precise, a P2P program implements several communication protocols and offers for each one of them the client and server aspects.

In CTA, the *retropropagation* service authorizes teacher and students to play a symmetric role. Consequently, their agent counterparts act as a client and / or a server. Thus, it was natural to choose a peer to peer architecture.

4.1 A Peer to Peer Architecture

Peer to peer systems are generally used by a large number of users to share files and they offer two main services 1) a file search service and 2) a file transfer service. Furthermore, to increase independence and anonymity of the participants the most sophisticated P2P

applications establish an additional level of interconnection called *overlay network* [23].

The *file search service* is one of the most delicate parts to design and implement because it determines the global performances of the application. Indeed, some P2P applications are used simultaneously by several tens of thousands of users and it is often complex to find a peer system able to provide the requested file.

The CTA *mobile agent layer* doesn't need a complex file search service because the history part of each IDO contains all information necessary to retrieve the other translations. Besides, the number of users is relatively small and the level of anonymity needed do not require an overlay network.

Broadly speaking and by analogy with classical peer to peer systems the *mobile agent layer* provides two services: a *contact management service* and a *file transfer service*.

4.2 Agent Architecture

The *agent* term refers to various concepts which resemble each other but do not correspond to only one definition. For example, in Computer Science, numerous fields of research use the notion of software agent either as concept, or as tool, or as both: artificial intelligence, artificial life, distributed systems and distributed applications, applications related to the new economy (e-applications), concurrent engineering, etc. Generally, an agent is considered as an entity possessing a certain number of properties very different in nature such as *intelligence*, *autonomy*, *mobility*, *proactivity*, *flexibility*, *sociability*, *perception* or *replication* [11], [12], [6]. We can notice that all the properties are not of the same nature (*autonomy* and *mobility* are two very different properties) and that they can be divided in two broad classes: *attributes* and *qualities* [20].

An *attribute* materializes a property of an agent, reduced to a mechanism, a software device perfectly known or with adaptable parameters. The main consequence of this is that an agent has or does not have an attribute. Mobility, replication and the perception of agents or the perception of sites are examples of attributes.

A *quality* can be simply defined as a behaviour, an aspect of an object or an entity that cannot be measured. Autonomy, intelligence, sociability, proactivity are examples of qualities. Qualities are difficult to measure as they are manifold. Consequently, there are various complementary models for a quality: for instance, there are several models of autonomy. There cannot be one definition of a quality.

In CTA we use the term *agent* 1) as a set of *attributes* and 2) as a *structuring unit*. Indeed, the aim of the application does not require to provide the agents with complex properties like qualities.

Attributes - For modularity and reusability we distinguish attributes closely related to the task assigned to the agent and attributes that are independent.

Three attributes were integrated into an agent: *mobility*, *replication* and *host perception*. These attributes and their implementation are independent of the task allocated to the agent.

Mobility allows an agent to migrate to others sites. Mobility is the most important attribute of our agents. Thus, this property will be described in the next section.

Replication allows an agent to create a clone of itself locally. Section 5 illustrates the use of local replication.

Host perception is an attribute necessary for the agents to move.

Structuring unit - A CTA software agent possesses a dual architecture (Figure 2):

- a sub-system which implements what concerns the achievement of the task assigned to the agent : the discovery, the analysis and the transfer of the IDO
- a sub-system which implements the three attributes mentioned above.

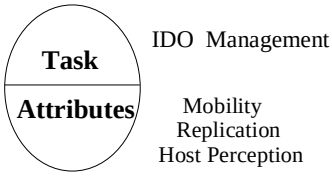


Figure 2 - Agent architecture.

4.3 Mobility Models

Mobility is a property which was particularly studied and implemented since the middle of the Nineties [25], [13], [15]. This interest was caused by the design of new applications using Internet and the inadequacy of the communication models used until then.

Indeed, at that time, the applications were structured on the remote code invocation model which used a great part of the bandwidth. To decrease the number of exchanged messages and to improve performances and the dynamic of the applications, White [25] showed that it was preferable that the client software agent moves on the site of the server entity. Thus, the client agent interacts locally with the server, then, when the treatment is finished it returns on its original site. Only the code of the agents is transmitted and not data of the server.

Today this mechanism is the center of many applications where a client software agent visits several sites in an independent way.

Mobility brings about interesting solutions to several problems met with the implementation of numerous applications [17]:

- a decrease of the network traffic,
- a use of common language or common execution environment, masking the heterogeneity of the platforms,
- a more natural modelling of entities occurring in applications,
- associated to a mechanism of agent replication, it improves fault tolerance.

We identified three mobility models which structure the architecture of a mobile agent application: *actual* or *virtual* mobility, *proactive* or *reactive* mobility and *strong* or *weak* mobility [2].

Actual mobility vs virtual mobility - There is a strong analogy between *actual mobility* and *message sending*:

- the message is sent to a remote host
- after a good reception by the receiver, the message is deleted on the local host.

In the *actual mobility*, all or parts of the mobile agent code are sent to the remote host and then removed from the local host. The advantage of the *actual mobility*, our choice, is that no specific agent component stays on a visited host.

In the *virtual mobility* a copy of the agent components is downloaded (and not transported) on the remote host and executed. There is a strong similarity between *remote code evaluation* and *virtual mobility*. The major difference is a greater independence of the remote code execution in *virtual mobility*.

Proactive mobility vs reactive mobility - In *proactive mobility* the agent asks when and where it wants to go. In *reactive mobility*, the local execution engine or the user decides when and where the agent must move. In the former case, agent mobility is also called *agent migration* by analogy with the *process migration* mechanism integrated in a few operating systems. In the latter case agent mobility will be called *user reactive mobility*.

Several mobile agent systems implement *proactive* and *user reactive mobility* such the Aglet system [16]: the Aglet ATP (Agent Transport Protocol) provides a *dispatch* request and a *retract* request.

Our mobile agent application uses only *proactive mobility*.

Strong mobility vs weak mobility - In *strong mobility* the execution context of the agent (mainly the stack and the program counter) must be captured and sent to the remote host. This one uses this execution context to resume the execution of the agent at the point where it stopped on the local host. The implementation of *strong mobility* requires the use of non standard execution engines (modified Java Virtual Machine or modified Tcl interpreter).

Weak mobility means that only code and data of the agent are moved on the remote host. A part of these data is used to restart the execution of the agent at a particular point of its code. Our mobile agents are structured according to the model of *weak mobility*. Contrary to *strong mobility*, *weak mobility* does not require any particular modification of the local and remote execution engines.

We are going to describe a scenario in a general way which will illustrate the dynamic brought by mobile agents.

5 Translation in Progress

The teacher publishes a text for translation. It is registered in a document called *ot* (*Original Text*). The history part of document *ot* is kept in a file *ot.hty*. When a student wants to download the file *ot* a **get** request is sent to the host known by the student which keeps this file. The service memorizes in the history of each file *ot* (*ot.hty*) all the **get** requests which it received. For each **get** request the user's IP address is recorded. We suppose for simplicity that there is only one user per host.

When a user creates a translation *tt* (*Translated Text*), the *change* service modifies the appropriate file *ot.hty*. Consequently, a file *ot.hty* contains two kinds of information: copies (**get**) and modifications (**change**).

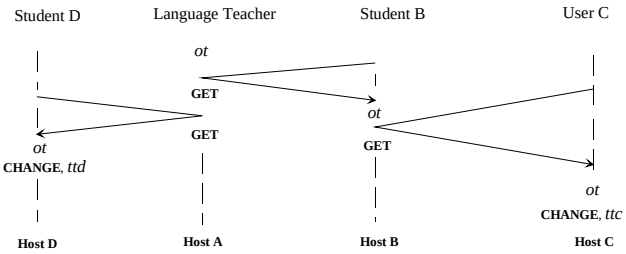


Figure 3 - A simple scenario.

In the scenario (Figure 3), *Student B* downloads file *ot* on *Host B* : the file *ot.hty* kept on *Host A* contains the line: *GET Host B*

The duplicated document *ot* on *host B* can also "receive" **get** requests. On *host C*, *user C* interested in the file *ot* present on the site of *Student B*, downloads the text for translation from the site *Host B* (file *ot.hty* on *Host B* contains the line: *GET Host C*) and publishes its own translation *ttc*.

The main consequence of the independent duplications of the file *ot* is that a teacher cannot know the location of all the copies of his document. To be able to build a map of the *ot* diffusion and to retrieve the translations suggested by the students, the language teacher uses mobile software agents. Figure 4 illustrates the work of mobile agents when the language teacher uses the *propagation* service.

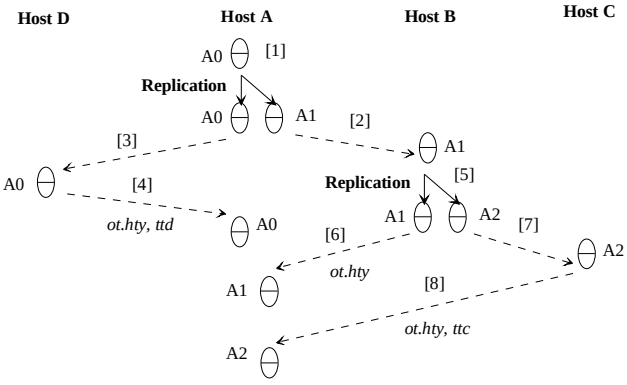


Figure 4 - *Propagation* service.

According to a policy previously defined (from events or at a regular interval of time), the language teacher launches a mobile software agent to analyse the local history of the document *ot*. The file *ot.hty* contains the set of users having requested a copy of *ot*: in our example, *Host A* sent a copy of *ot* to *Host B* and *Host D*. The software agent analyses the *ot.hty* file and duplicates itself as necessary. Each clone (A_0 and A_1) goes towards a different implied host. When clone A_1 arrives on *Host B* **[2]**, it analyses the local file *ot.hty*, creates a clone A_2 which goes to *Host C* **[7]** and returns to *Host A* with the local history file *ot.hty* **[6]**. When clone A_2 arrives to *Host C*, it finds a translation *ttc* and returns to *Host A* with the local files *ot.hty* and *ttc* **[8]**.

With the different *ot.hty* files, the language teacher is able to build a map of the *ot* file propagation and to retrieve the translations scattered in the network. This process is completely asynchronous.

There are a few differences between the executions of *retropropagation* service and *propagation* service. When *User C* starts the *retropropagation* service the mobile agent analyses the local file *ot.hty* and navigates according to the contents of the remote files *ot.hty* which have been read (Figure 5). Students are supposed to be interested only in the translations and not in the history parts of the IDO. Thus, only *ttd* is downloaded on *Host C*.

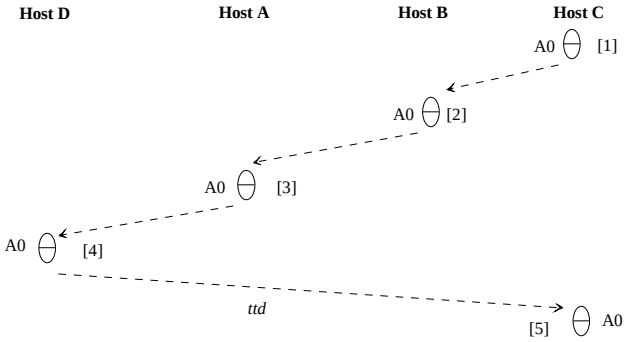


Figure 5 : *Retropropagation* service used by *User C*.

6 CTA Software

6.1 User Interface

We chose Zope [26] as CTA *user interface*. Zope is an open source web application server created by Digital Creations. In addition to a web server, Zope provides many tools for content management and information publishing on the web: it integrates its own database compatible with the SQL language and its own programming language.

Applications designed with Zope have an Internet oriented interface, i.e. all the possible actions are accessible via a simple Internet browser. This implies the use of a language interpretable by these browsers. The HTML (HyperText Markup Language) is sufficient to present static information, but in our application the information shown differ in every execution. Thus, the HTML is not appropriate. Zope makes it possible to create dynamic web pages with its own specific language: the DTML (Document Template Markup Language). The DTML is closely related to the HTML from which it derives. The DTML makes it possible to directly generate interpretable HTML code by any browser.

Zope is a multiuser software. That means that a Zope administrator can create users each one having a name and a password, groups of users, and to give them specific permissions.

Zope is essentially written in Python [19], a powerful object oriented scripting language. Consequently another strong point of Zope is its great compatibility with many operating systems and the developed applications are easily portable when the system is changed.

Lastly, when Zope shows its limits, the use of external methods written in Python by the programmer himself, makes it possible to provide the applications with innovating functionalities and impossible to complete with a Zope environment only. In the CTA application, this one does not allow to resolve all the problems and to make all the treatments which one would want. That is why a supplementary software layer totally written in Python was added.

6.2 Mobile Agent Layer

During the execution of the application, the user who directs the progress of the collaborative work, i.e. the teacher, can at any time start a propagation. That means that he will be able to retrieve documents located on the machines of the other users of the application, i.e. the students, thanks to mobile agents. On each machine (including the teacher's machine) where the application is launched, a server is running and waits for the connection requests. When the teacher starts the propagation, a mobile agent is created and will migrate from host to host, from server to server, so as to fill its task.

Eventually, the propagation is transparent for all the users of the application including the teacher who has only to press on a button to launch this functionality.

The retropropagation has a functioning similar to the propagation, the main difference being situated at the level of its activation place. Indeed, all the users except the teacher can activate this function The aim is the same, i.e. retrieving the documents from the machines participating in the application.

As well as propagation, retropropagation is transparent to all the users.

6.3 IDO Implementation

The application makes the distinction between three sorts of documents: *user documents*, *propagated documents* and *history files*.

User document is to be understood as a document downloaded from a machine or modified by the user on a machine. The download is explicitly done by the user. At the start of the application a user document is created by the teacher. Once registered, this document becomes accessible to all the students who can download it on their own machines. When it is downloaded on a machine, an associated history file is then created. It is the user document which circulates when (retro) propagation takes place between the different machines.

A *propagated document* is a document which is brought by a mobile agent from a remote machine. These documents are stored in a special directory. The name of a propagated document contains the IP address of the machine it comes from or the name of the user it belongs to.

A *history file* is associated to a user document. This file has an '.hty' extension (Section 4) and its name is similar to the user document it corresponds to. For example, for a user document called ''Appli'' the associated history file will be called ''Appli.hty''. This file contains the events that occurred to the user document.

Three kinds of events can be found in a history file (Figure 6):

- a user document is downloaded from a remote machine. Consequently, a history file is created and the PARENT event is added into it.

The PARENT event has one argument: the IP address of the source machine from which the user document comes.

Here is an example of a full PARENT line:

PARENT 192.168.2.30

This line means that the user document has been downloaded from the machine which IP address is 192.168.2.30.

It will be noticed that one and only one PARENT line can be found in a history file because a user document can only come from one place

- the user document is modified by the user (**change** service).

The MODIFIED event is then added to the history.

The MODIFIED event has one argument: the IP address of the current machine.

Here is an example of a full MODIFIED line:

MODIFIED 192.168.2.32

The user document has been modified on the machine which IP address is 192.168.2.32.

- a remote machine has just downloaded the user document that is on our machine (**get** service).

The GET event is then added to the history.

The GET event has one argument: the IP address of the machine which has downloaded the user document.

Here is an example of a full GET line:

GET 192.168.2.29

The user document has been downloaded by the machine which IP address is 192.168.2.29.

```
PARENT 192.168.2.30
MODIFIED 192.168.2.32
GET 192.168.2.29
MODIFIED 192.168.2.32
```

Figure 6 - History file.

Figure 6 shows an example of a complete history file with all the lines presented above. The user (a student) is working on the machine which IP address is 192.168.2.32.

The file containing the text to be translated has been downloaded from machine 192.168.2.30 (1st line). The file has been modified by the student (2nd line), then the original document (and not the student's translation) has been downloaded by a student on machine 192.168.2.29. Finally, the last line of the history file indicates that the local student has changed his translation.

6.4 Using CTA

For a user (teacher or student) to use CTA, it is necessary that

- the daemons of the Zope application are running,
- the user is registered by Zope server.

Then, the user has access to the home page of the application by means of a browser by indicating the appropriate URL, for example:

http://localhost:8080/Translation

The user gets the screen shown on Figure 7.

The interface is as simple as possible so that it is perfectly clear and intuitive even for an occasional user. Working with the CTA software is reduced to a simple local navigation to make the application accessible to most users. This simplicity was the objective followed since its design (cf. Section 2.2).

Figure 7 - CTA home page.

The user must log as teacher (first button) or student (second button) with a password.

If the user logs as teacher he gets the screen shown on Figure 8.

Figure 8 - Teacher home page.

He gets a window where he will insert the text to be translated. Then he will indicate the name of the document. This name will be visible for students.

Three buttons are available:

- one to start *propagation*, which allows him to retrieve the different translations accessible
- one to consult the list of the retrieved translations (Figure 9)
- a reset button.

After retrieval, the teacher gets the translations suggested by the students participating in the application (Figure 9). The teacher's screen also displays the names of the participants. There is no anonymity for the teacher. He can check the translation of a particular student by clicking on the corresponding document name.

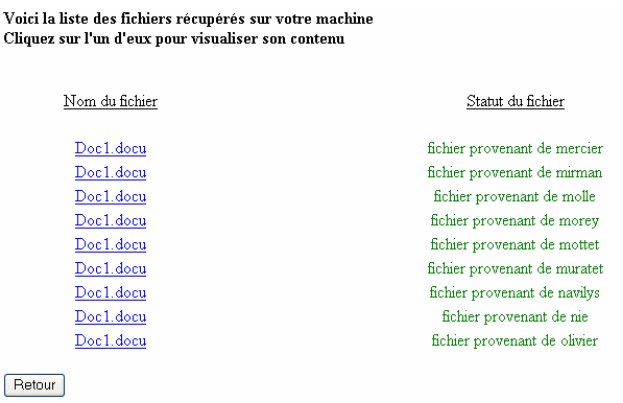


Figure 9 - After a propagation.

If the user logs as student he gets a different screen (Figure 10).

Figure 10 : Student home page.

The student must establish a connection to a Zope server before where a user document is stored, i.e. the text to be translated. This server can be the teacher's server or another machine. When the connection is established, the name of the user document appears and by clicking on this name, the user retrieves this document.

When the student presses the *Translation* button a window appears with the text to be translated. The student can start his own translation and saved it.

The next button launches the *retropropagation* and its action is to retrieve the other students' translations already available.

The last button (*Consult)* allows the student to consult these translations (Figure 11). The anonymity is kept: only the IP address is displayed.

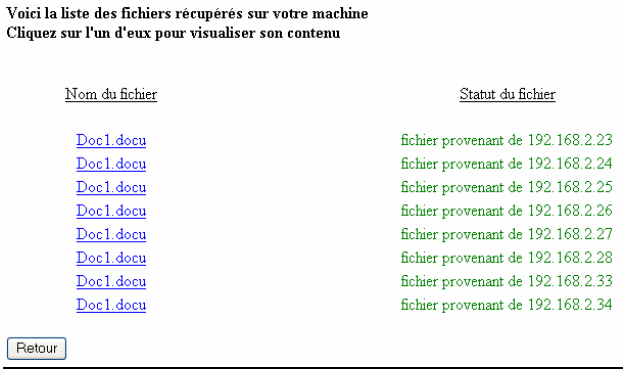


Figure 11 - After a retropropagation.

7 Concluding Remarks

The use in situation of the CTA prototype allowed us to learn the following lessons: when only the *propagation* service is used in the application, *actual mobility* is perfectly adapted. No code relating to the

mobile agents remains present on the computer systems of the students.

On the other hand, when the *propagation* and *retropropagation* services are jointly used, each site has to possess the mobile agent part and the agent server part, which makes it less interesting to delete the code of the agents having finished their execution. Consequently, *effective mobility* is less efficient.

In addition, the prototype suffers from limitations intrinsic to certain technical and conceptual choices: the use of mobile agents introduces security problems not completely solved and the peer to peer architecture is sensitive to the volatility of participating computer systems.

A significant improvement of the prototype would be to introduce additional mechanisms for a better managing of this volatility.

The CTA implementation revealed several positive aspects. The using, the modelling and the implementing of the IDO concept gave us complete satisfaction. This concept resulting from Concurrent Engineering was easily adaptable to an application belonging to a different field. Programming by mobile agents allowed an elegant implementation of the asynchronous mechanisms necessary to an application which had to be as light and transparent as possible.

7.1 Acknowledgement

I would like to thank G. Vayssie and O. Peron for their contribution to the development of the CAT Software and the English teachers H. Lamouliatte and O. Depitre.

8 References

[1] D.P. Anderson, J. Cobb, E. Korpella, M. Lebofisky, D. Werthimer, "*SETI@home: An Experiment in Public-Resource Computing*", **Communication of the ACM**, Vol. 45, No 11, November 2002, pp. 56-61.

[2] G. Cabri, L. Leonardi, F. Zambonelli, "*Weak and Strong Mobility in Mobile Agent Applications*", 2nd International Conference and Exhibition on the Practical Application of Java (PAJAVA 2000), Manchester (UK), April 2000.

[3] D.E Carter, B. Stilwell Baker, **Concurrent Engineering, The Product Development Environment for the 1990s**, Addison-Wesley, 1991.

[4] K. Curran, "Peer-to-peer networking Collaboration Within Education", **Journal of Educational Multimedia and Hypermedia** (2002) 11 (1), pp 21-30.

[5] *http://edutella.jxta.org*

[6] O. Etzioni, D. S. Weld, "*Intelligent agents on the Internet: Fact, Fiction, and Forecast*", **IEEE Expert** 10(4): pp 44-49, 1995.

[7] A. Fuggetta, G. P. Picco, G. Vigna, "*Understanding Code mobility*", **IEEE Transactions on Software Engineering**, Vol. 24, No 5, May 1998, pp. 352-361.

[8] L. Gong, "*JXTA: A network Programming Environment*", **IEEE Internet Computing**, Vol. 5, No 3, May-June 2001, pp. 88-95.

[9] *http://www.groove.net*

[10] A. Jeantet, J.F. Boujut, **Conception de produits mécaniques**, M. Tollenaeree, chap. 5, Paris, Hermes, 1998.

[11] N. Jennings, M. Wooldridge, "*Software Agents*", **IEEE Review**, January 1996, pp 17-20.

[12] N. Jennings, K. Sycara, M. Wooldridge, "*A Roadmap of Agent Research and Development*" in **Autonomous Agents and Multi-Agent Systems**, 1, pp 275-306, Kluwer Academic Publishers, Boston, 1998.

[13] D. Johansen, R. van Renesse, F. B. Schneider, "*Operating system support for mobile agents*", Proceedings of the 5th Workshop on Hot Topics in Operating systems, May 1995, pp. 42-45.

[14] G. Kan, "*Gnutella*", in Andy Oram ed., **Peer to Peer: Harnessing the Power of Disruptive technologies**, pp 94-122, O'Reilly, 2001.

[15] D. Kotz, R. Gray, S. Nog, D. Rus, S. Chawla, G. Cybenko, "*AGENT TCL: Targeting the Needs of Mobile Computers*", **IEEE Internet Computing**, Vol. 1, No 4, July-August 1997, pp. 58-67.

[16] D.B. Lange, M. Oshima, **Programming and Deploying Java Mobile Agents with Aglets**, Addison-Wesley, 1998.

[17] D.B Lange, M. Oshima, "*Seven Good Reasons for Mobile Agents*", **Communication of the ACM**, Vol. 42, No 3, March 1999, pp. 88-89.

[18] W. Nejdl and all, " *EDUTELLA: A P2P Networking Infrastructure Based on RDF*", WWW2002, May 7-11, 2002, Honolulu, Hawaii, USA.

[19] *http://www.python.org*

[20] E. Sanchis, "*Designing new Agent Based Applications Architectures with the AGP Methodology*", WETICE, 1st International Workshop on Theory and Practice of Open Computational Systems (TAPOCS 2003), June 9-11, 2003, Linz, Autriche

[21] J.L. Selves, E. Sanchis, Z. Y. Pan, "*New products development within a Concurrent Engineering environment Knowledge and Software Tools*", IEPM'01, Proceedings for the International Conference on Industrial Engineering and

Production Management, Quebec City (Canada), August 20-23, 2001.

[22] C. Shirky, "*Listening to Napster*", in Andy Oram ed., **Peer to Peer: Harnessing the Power of Disruptive technologies**, pp 21-37, O'Reilly, 2001.

[23] J. Touch, "*Overlay networks*", **Computer networks**, 3, (2-3), 2001, pp. 115-116.

[24] A. Ward, J.K. Liker, J.J. Cristiano, K. Duward, Sobek, II, "*The second Toyota Paradox: How Delaying Decision Can Make Better Car Faster*", **Sloan Management Review**, Spring 1995, pp. 43-41.

[25] J.E White, "*Telescript Technology: Mobile Agents*", General Magic White Paper, in Bradshaw J. **Software agents**, AAAI/MIT Press, 1996.

[26] *http://www.zope.org*

Human-Agent Interaction: Case Studies in Human Supervised UAV

Henry Hexmoor and Srinivas Battula
Computer Science and Computer Engineering, University of Arkansas,
Fayetteville, AR 72701, USA.
{Hexmoor, sbattul} @uark.edu

Keywords: Agents, Autonomy, Trust, Collaboration, Help

Received: August 15, 2003

In this paper we offer case studies of empowering agents with adjustment of cognitive notions of autonomy and trust that enable them to have a more socially adept interaction with a human supervisor. The application domain is control of unmanned aerial vehicles. Agents learn to change autonomies as they observe they learn the relationship between their assumed autonomy and performance. Agents also learn to change their reliance on human supervision as it takes different lengths of time.

1 Introduction

Human-Agent Interaction (HAI) is an emerging area of research in agent-based systems. In general, agent-based computing has been beneficial in four areas and human-agent interaction is found in all of these areas. First, agents are used in automation of dirty, dull, and dangerous as well as tedious, boring, and routine tasks to relieve humans of such duties [6]. Examples in this area can be found in agents embodied in desktop assistants [8] or intelligent in service of humans. Human supervisors benefit greatly from delegating tasks to such agents [5]. Secondly, agents are used to produce an improved human sense of "presence" for humans collaborating in physically disparate locations. Examples in this area are found in agents in knowledge management tasks like war-rooms and human users benefit from agents who proxy for their human counterparts. Third, agents can be used in democratisation of computing, services, and support. Examples in this area are agents in municipal functions such as the department of motor vehicles or public libraries and virtual museums. Here, the public enjoys the benefits of agent services. Fourth, agents are used in reduction of redundancy and overlap due to competition. Research in this area can increase collaboration between agent collectives such as in institutions, organizations, and teams. Examples in this area are found in agents that facilitate tracking and sharing power or telecommunication services. In human-agent interaction, agents might be cognitive assistants capable of discovering human preferences, personality, and emotions. With this, agents will gain human trust along with permission to assure increased autonomy while providing greater human control. Agents will also form social networks that will facilitate their greater ability to work together and to collaborate. We envision agents that will be socially adept. This contributes to robustness and adaptability of collaborative enterprises. Theories and models of human-agent interaction are needed as part of collaborative enterprises to provide foundations for constructing systems able to work with each other and with the people using them.

In complex tasks where humans and agents share control and both make decisions, human-agent interaction must accommodate mixed initiatives. Agents that allow human intervention in their actions are said to have adjustable autonomy [1, 2, 4]. Here an agent's autonomy can be varied dynamically. Adjustable autonomy holds the promise of lowering human controller's burden of continuously controlling the agent and alleviates the agent from dependence on the human controller. The delays involved in making a decision on behalf of the agent and conveying it to the agent may decrease the agents' performance. Time-critical systems cannot afford this delay and in some cases a medium quality decision made by an agent in time will be superior to a high quality decision by a human-controller which may not be timely. However, there are situations where not all decision-making autonomy can be given to an agent based on the assumption that an agent makes good decisions.

Trust is an important social notion between humans and agents. Humans must trust agents they supervise and much needs to be in human acceptance and trust on agent decisions. In this paper we will explore the inverse issue, that of trust as a form of agent reliance on quality and timeliness of human input. We also present reliance as a form of granting power to the human controller.

In mixed teams of humans and agents, interactions that require agents and humans issuing request form one another must be designed to account for human cognitive factors. Agent request issued to a human must account for contextual connotations of the request. This was similar to the goal of DARPA's pilot's associate program and the more recent MICA program. Large-scale teams of semi-autonomous vehicles were intended to be controlled by a relatively small number of human operators. In this paper we report on a machine learning scheme for asking for helping from a human when agent-controlled UCAV pilots need assistance from a human piloted UCAV. Each situation

requires a different request phrase. The phrases embody the context of the situation and the human interpretation of the request.

In the rest of this paper we will begin by describing our implemented testbed that we used to experiment with adjustable autonomy and human-agent interaction [9]. We will then present a few empirical results and end with concluding remarks.

2 Testbed

Our simulator consists of a mountainous terrain with SAM's and a number of UCAV's that fly over them [3]. All the UCAV's are partially autonomous agents whose autonomy can be adjusted by the human controller dynamically. UCAV's starting from the base, fly over the mountainous terrain to reach their destination. UCAV's and SAM's have a visible region within which they can attack one another. *Hit probability* is the probability of the UCAV being hit by the SAM's. If the hit probability of a UCAV crosses a certain limit the UCAV's are considered to be shot-down by the SAM's and they disappear from the simulator. In addition, if two UCAV's are *in coalition* the hit-probability of both the UCAV's decreases by a factor due to the confusion of the SAM(s). A SAM tries to hit the plane as soon as it enters its visible region. When a UCAV comes across a SAM in its course to reach the destination it may *Start Avoiding* the SAM by itself or may ask help from other agents to attack the SAM. The UCAV initially tries to avoid the SAM until its hit probability crosses a certain limit. When the hit probability crosses a limit, it requests for help from other agents. Hit probability of an UCAV is proportional to the number of SAM's that have UCAV in their Visible region. Each Agent gets its turn sequentially. A *cycle* is completed when all the agents get their turn once. The *cycle* continues until all the agents reach their destination [10].

The following are the 10 states that govern the behaviour of an agent in the simulator:

1. *Fly to Target*: This is a default state. In this state the agent's goal is just to reach the destination
2. *See SAM*: The agent enters this state as soon as it sites a SAM in its visible region. In this state the agent reasons whether to avoid SAM by itself or to seek help.
3. *Start Avoiding*: The agent enters this state when it is in the visible region of a SAM and can avoid the SAM by itself.
4. *Waiting for Help*: The agent enters this state if the hit-probability crosses a certain limit and its better to seek help from other agents than avoid by itself. In this state the agent waits for help from other agent.
5. *Offering Help*: The agent enters this state when any other agent of the system is waiting for help. In this state the agent offers help, which may be accepted or rejected by the help-needing agent.
6. *Being Helped*: The agent enters this state when one of the agents has agreed to offer help and it is willing to accept it.
7. *Helping*: The agent enters this state when another agent accepted help offered by this agent and it is on its way to help the agent.
8. *Helping and See SAM*: The agent enters this state when it is helping another agent and on its way sites a SAM itself.
9. *Helping and Avoiding SAM*: This state is a result state to the previous one.
10. *In Coalition*: When an agent helps another agent the helping agent and the helped agent form a Coalition. They break the Coalition only after both of them are out of any of the SAMs visible region. The hit-probabilities of the planes decrease considerably with the two agents in coalition.

The following are the set of permissions that an agent requires to operate completely autonomous in the system. *Attack SAM*: The agent needs to have this permission set to attack a SAM as soon as it sites it.

1. *Attack SAM*: The agent needs to have this permission set to attack a SAM as soon as it sites it.
2. *Avoid SAM*: The agent needs to have this permission set to avoid SAM by itself. An agent that doesn't have permission to avoid enters *Waiting for Help* state
3. *Get Help*: The agent needs this permission set to accept help offered by other agent. An agent that doesn't have this permission has no choice of selecting the helping agent.
4. *Offer Help*: The agent needs this permission set to offer help to other help-needing agents.
5. *Help*: The agent needs to have this permission set to help other agents when another agent accepts it

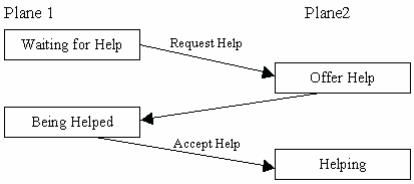


Figure 1 Help Scenario

Human controller sets the permissions of an agent initially when an agent is created. The more permission's the agent has the more autonomous is the agent. For example, if an agent wants to attack a SAM and doesn't have the permission to perform the action it has to get permission from the human controller. Human controller can give the agent permission to attack, or deny permission. In addition, if an agent is given all the

permissions and later if the human agent wants to change it he can do so by changing the agent's autonomy dynamically.

When an agent asks for permission from the human controller to act in a particular situation the human controller has to make quick and wise decisions that improve the system performance. The human controller can be in two states *Busy* (i.e., responding to another agent) or *Idle*. The agent has to wait until the human controller makes a decision and conveys the decision to it. In our simulator the overall system performance increases with the decrease in average hit-probability of the agents. When an agent enters the Visible region of a SAM the hit-probability increases with time until the agent gets out of the visible region by avoiding SAM or another agent comes to rescue. The agent's hit-probability at each cycle is recorded. The hit-probability of an agent can be as low as 0.0 when it is not in visible region of any of the SAM's or as high as 0.8, which we set as a higher limit on hit-probability. Agent's which have hit-probability of greater than 0.8 are considered shot down by the SAM's.

Cumulative Hit-Probability (CHP) of an agent is the sum of hit-probabilities of the agent in each cycle through out the simulation run (i.e. from the time the agents took off from the base until they reach their destination) divided by the number of cycles the agent has hit-probability greater than 0.0. *Average Hit-Probability* (AHP) is the average of the cumulative hit-probabilities of each agent.

$$\text{CHP} = \frac{\sum_{i=1}^{l} p_i}{n} \qquad \text{AHP} = \frac{\sum_{j=1}^{m} chp_j}{m}$$

- "p_i" is Hit-Probability of agent at cycle i
- "n" is the number of cycles in which hit-probability of the agent is greater than 0.0
- "l" is the total number of cycles in a simulation run
- "chp_j" is cumulative Hit-Probability of agent j
- "m" is the number of agents

Agents that require permission from the human controller add permission message to a queue from which the controller first selects it and gives the permission or denies the permission. There is a delay involved from the time the agent adds the permission message to the queue and gets a response from the human controller. The factors that affect the delay are

1. The number of permissions asked by other agents
2. The human computer interaction system
3. The efficiency of the human controller (we ignored this in our simulator)

Here some important questions arise.

1. How long should the agent wait for the human permission
2. Should the agent take over control and make an autonomous decision.

The permissions given by the human controller are recorded in the agent's history together with the hit-probability of the agent when the permission is given and cycle number. Other agents use these permissions when they are in a similar situation. Two agents are considered to be in a similar situation if they require the same kind of permission, the permission given is not more than 30 cycles old and the hit-probabilities are close. Permissions given to an agent that are 30 cycles old become invalid.

The delay in communication between the human controller and the agent deters the performance of the system. With the increase in number of agents the number of decision's to be given by the human controller increases and this degrades the performance of the system further. So we have set an upper limit on the waiting time beyond which further waiting of the agent has degrading effects.

To make a decision autonomously the agent needs to know:

- when should it Start Avoiding,
- when should it Wait for help,
- when should it offer help,
- when should it accept help,
- when should it help

The predefined rules set to the above actions are:

- An agent starts Avoiding when it sites a SAM and its hit-probability <= 0.2.
- An agent waits for help when it is avoiding a SAM and its hit-probability > 0.2.
- An agent offers help when it is in *Fly to Target* state and some other agent needs help.
- An agent, which is waiting for help, accepts help if another agent offers it.
- An agent helps another agent that accepted its help offer.

After waiting for a maximum time limit the agent chooses to follow one of these rules. The agent's decision may not be convincing in all situations. In those cases the human controller can interrupt the agent and gives his decision to the agent.

3 Experiments and Results

In this section, we will discuss results for adjustable autonomy as well as trust. We begin with four adjustable autonomy scenarios discussed.

3.1 Adjustable Autonomy

The x-axis in Figures 2 to 5 represents the number of agents taking part in a simulation run and the y-axis represents the hit-probability averaged over of all agents in the scenario. Figure 2 shows the average hit-probabilities of 2, 3 and 4 agents when all the decisions have been autonomous, i.e., without human control.

Permissions required to make autonomous decisions by an agent are given to each agent. From the figure we can observe that the average hit-probability remains almost constant. All agents follow a predefined rule set in making a decision.

Figure 3 shows the average hit-probabilities of 2, 3 and 4 agents when the human controller makes all the decisions. i.e., the agents are completely controlled by the Human. We observe that the average hit-probability increases with the increase in number of agents. With increased number of agents, the human controller is flooded with more requests from agents for permissions. Therefore, the delay in making a decision increases the hit-probability for the waiting agents. The increase in hit-probability is more between 3 and 4 agents than between 2 and 3 agents. Figures 4 and 5 shows changes in average hit-probability when the agent's autonomy can be adjusted dynamically. We have considered two cases in which the autonomy of the agent will be varied dynamically. In the first scenario, an agent doesn't wait for the human controller's decision and makes an autonomous decision based on the rule set and continues with it. However, if the human controller feels that the agent did not make a wise decision he can override it with his decision and ask the agent to proceed according to the new decision.

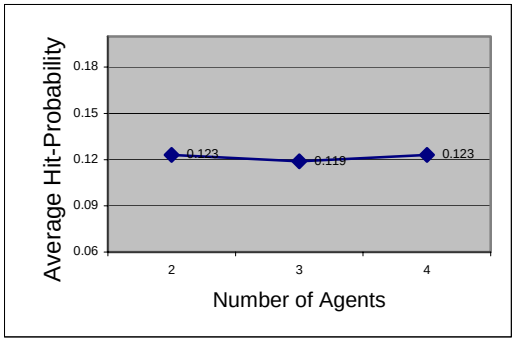


Figure 2. Agent Controlled

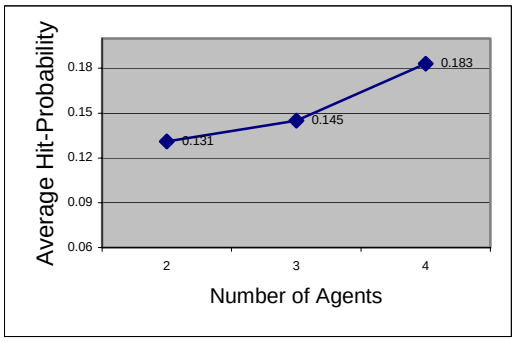


Figure 3. Human Controlled

Figure 4 shows the average hit-Probabilities of 2, 3 and 4 agents.

The second scenario is where the state of an agent can't be changed or a decision made once can't be reverted. Here we have set an upper limit on the number of waiting cycles the agent waits before making an autonomous decision. If the human controller feels that the agents can be given more autonomy he can decrease or increase the waiting cycles. In figure 5 we observe that the hit-probability increases with increase in number of agents but this increase is considerably less than the increase in figure 3 in which the human controller makes all the decisions.

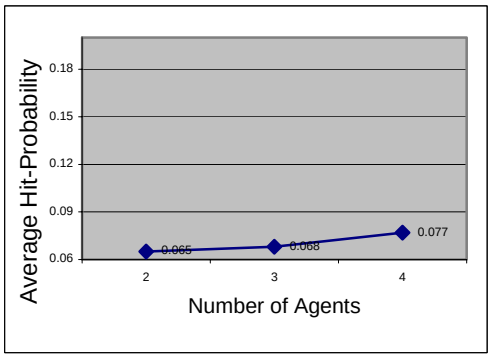


Figure 4. Autonomous Decision made by Agent, which can be reverted by human controller later.

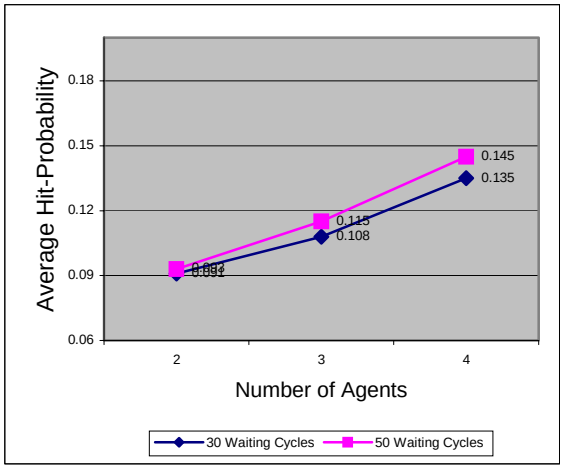


Figure 5. Autonomous Decision made by agent after waiting for certain time limit, which can't be reverted by human controller.

Reasoning about *reliance* as a form of trust between human and agent is another method to manage adjustable autonomy where the human controller has the most control over the agents. This is presented next.

3.2 Reliance

Let's consider that the human controller doesn't have control over the agents but all the agents presume that a human controller's decision is superior to a decision made by them. To reiterate, the delay involved in giving a decision increases with the number of requests, agents cannot wait for the human's decision beyond a certain point, which may increase their hit-probabilities. Agents conceive of a Global Human Reliance Value (GHRV) for

human decisions with a maximum value of 5 [11]. GHRV gives a measure of the degree of trust agents has over the human controller [12]. The reliance increases if the human controller responds to a request and decreases if the human controller fails to respond to a request in time (say 40 waiting cycles). Agents wait for human controller's decision for a certain period of waiting cycles based on the reliance value and on the social power authority that we discuss below. If the reliance value is low, agents wait for a small period before making an autonomous decision. Social power is a direct consequence of reliance. Power exercised by Human controller on agents is of two forms:

1. Authority leads to power, which is exercised by an authority.
2. Expert knowledge leads to power gained by expert solutions and suggestions.

In our simulator agent's waiting time is governed by the following equation:

Maximum_Waiting_Cycles = I + GHRV * 5

Where I is the number of waiting cycles agent waits initially when it has no reliance on Human controller. Therefore, each agent waits for at least I waiting cycles before making an autonomous decision even when the reliance value is 0. The value of I is dependent on Authority form of power. With more authoritative power agents wait more time before making an autonomous decision. Power of expertise is directly proportional to GHRV. When human controller gives a decision the value of GHRV is incremented by 1. So in cases when a human controller is flooded with requests he fails to respond to some of the requests, which decreases the GHRV. With less GHRV agents that require permission wait less before making an autonomous decision. This considerably decreases the number of waiting cycles agents wait for a permission and unsuccessful in getting the human controller's response. Human controller will be less effective in influencing the agents with the decrease in GHRV. Figure 5 shows the effect of authoritative power on the hit-probability. With more authority agent's ability to make an autonomous decision is narrowed and hit-probability increases with the increase in number of agents as human controller fails to respond to all the requests in time.

As we have already mentioned human controller's decision is stored in the agents' history. Two agents in a similar situation can use the same response of the human controller. In addition agents also have Agent-Agent reliance values among them. Each agent's reliance on other agents is maintained in an Agent-Agent Reliance Value (AARV) array. Agent-Agent reliance value also varies between 0 and 5. An agent first interacts with other agent it relies on most to check if that agent has received a response from the human controller to perform the same action. The other agent responds to the agent's request as follows

1. Returns 0 if the human controller has not given permission to perform the action
2. Returns 1 if the human controller has given permission to perform the action
3. Returns 2 if it couldn't find it in its history

Reliance value increases by 1 in the first two cases where the agent returns 0 or 1. It decreases by 1 if the agent returns 2 i.e. agents rely more on an agent that provides them with information that is useful in making an autonomous decision. The following figures illustrate the number of interactions between agents and the number of human-agent interactions and how average hit-Probability is affected when Agent-Agent and Human-Agent reliance is considered. In figure 6 we can observe that the average hit-probability increases with increase in the number of agents. The average hit-probabilities in this case are very much similar to the average hit-probabilities of figure 5. Figures 7 and 8 give the number of interactions between human-agent and agent-agent. With increase in the number of agents the interactions between agents increase rapidly, however there is not much increase in the human-agent interactions.

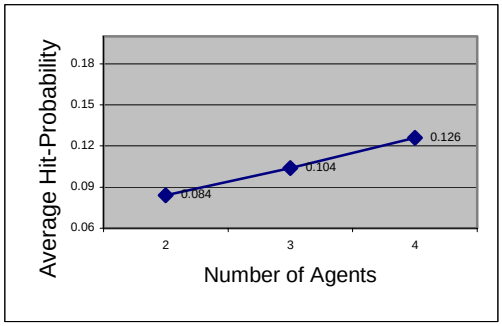


Figure 6 Average Hit-Probabilities of agents when agent-agent and human-agent trust is considered

3.3 HELP

This section evaluates the effectiveness of agents help requests on other agents. In some systems helping agents incur cost on the receiving agent for the help given by them. Altruist and benevolent agents for example help others without incurring any costs. In such a system where there are no costs incurred on help-receiving agents the following factors effect cooperation between agents [7].

1 Social normative factors
2 Emotional factors
3 Strategic factors.

Human Controller gives feedback for each request by the agent based on the accuracy of the request made in a particular situation. Some agents require help with greater degree than other agents.

The preferences are based on:

1. The state of the agent receiving help e.g.: if two agents have hit probabilities 0.1 and 0.5 then the

helping agent is going to help the agent with higher Hit-probability.
2. The risk involved in helping an agent e.g.: if helping an agent puts this agent into risk then the agent may avoid helping such an agent
3. The relation-ship with the receiving agent e.g.: the agent's relationship with the helping agent.

Five different situations where agents seek help from other agents are considered. Human controller gives feedback for each request made by agents in a particular situation.

The five situations are:
Situation 1 - Agent doesn't see SAM and its not attacked
Situation 2 -- Agent doesn't see SAM but attacked
Situation 3 - Agent sees SAM and being attacked
Situation 4 - Agent sees SAM and attacked closely
Situation 5 - Agent attacked very closely

The five different help requests considered are:
1. Mayday
2. I want help
3. I need help
4. I may need help
5. Can I get assistance?

A 'Mayday' request needs immediate attention than a 'Can I get assistance?' request. The feedback given by the Human controller is recorded in a two-dimensional feedback array. Feedback is given on a scale of -10 to +10. Figure 2 shows a feedback array generated during a simulation run.

Feedback given by Human Controller for Situation3 and request "I need help" is +6, which is highest in the row. The next time an agents comes across Situation3 it announces the request "I need help" as it received the best feedback. Similarly, when a situation similar to any of the above five arises the help-needing agents announce a request that got the best feedback.

4 CONCLUSION

The aim of human-agent interaction is to design interfaces and cognitive approaches that increase access of human and agent over one another's decision making process. In this paper we have presented results of experimentations with endowing agents with social abilities. A few tradeoffs are shown in adjustable autonomy of agents. Here we observe that level of human control can be increased while preserving agent performance. Dynamic adjustment of agent wait cycles for a human decision as well as experience an agent gains from waiting are two specific methods we have explored. We defined a form of trust between agents and humans we called reliance. With this we showed how agents may reason over timeliness and significance of human guidance. We also showed how agents must find appropriate phrases when requesting ask from their human counterpart. The results are promising and show the way to similar methods.

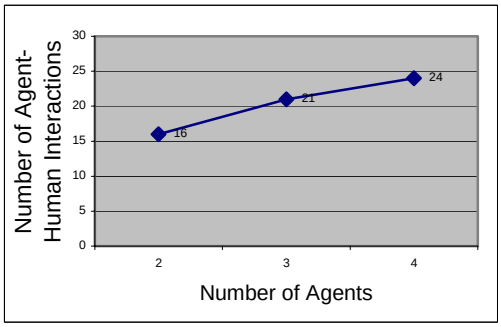


Figure 7 Number of interactions between human controller and agent.

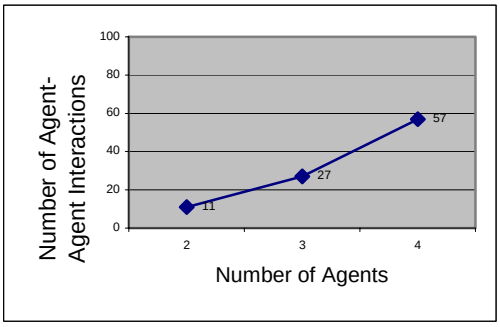


Figure 8 Number of agent-agent interactions

	Mayday	I want help	I need help	I may need help	Can I get Assistance
Situation1	-8.0	-6.0	-2.0	2.0	7.0
Situation2	-7.0	-2.0	2.0	7.0	3.0
Situation3	-1.0	2.0	6.0	1.0	-5.0
Situation4	2.0	8.0	3.0	-1.0	-6.0
Situation5	9.0	6.0	0.0	-2.0	-8.0

5 Acknowledgements

This work is supported by AFOSR grant F49620-00-1-0302.

References

[1] G.A. Dorais, R.P Bonasso, D. Kortenkamp, B. Pell, and D. Schreckenghost, 1998. "Adjustable Autonomy for Human-Centered Autonomous Systems on Mars", In Proceedings of the First International Conference of the Mars Society.

[2] M. A. Goodrich, D. R. Olsen jr., J. W. Crandall, and T. J. Palmer, 2001. "Experiments in Adjustable Autonomy". In Proceedings of the IJCAI-01 Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents.

[3] H. Hexmoor, and X. Zhang, 2002. "Socially Intelligent Air Combat Simulator", In Proceedings of The Seventh Pacific Rim International Conference on Artificial Intelligence, (PRICAI-02), Tokyo, Japan.

[4] P. Scerri, D. Pynadath, and M. Tambe, 2002. "Towards adjustable autonomy for the real-world". Journal of AI Research (JAIR), Volume 17, Pages 171-228.

[5] D. Shapiro, 2001. "Value-driven agents", Ph.D. thesis, Stanford University, Department of Management Science and Engineering.

[6] P. Maes, 1994. "Agents that Reduce Work and Information Overload". Commun. ACM 37,7, 31-40

[7] P.Rizzo,A.Cesta and Maria Miceli,1995. On Helping Behavior in Cooperative Environments, *Published in Proceedings of the International Workshop on the Design of Coopertive Systems, pp 96-108*

[8] M. Huhns and M. Singh, 1998. "Personal Assistants," In IEEE Internet Computing, Vol. 2, No. 5: Sept-Oct 1998, pp. 90-92, IEEE press.

[9] P. Scerri and N. Reed, 2001. "Designing Agents for Systems with Adjustable Autonomy". The IJCAI-01 Workshop on Autonomy, Delegation, and Control: Interacting with Autonomous Agents.

[10] H. Hexmoor and S. Battula ,2003. "Towards Collaboration between Human and Social Agents that mind Human Social Personality". International Symposium on Collaborative Technologies and Systems. (to appear)

[11] S. Marsh, 1992. "Trust and reliance in multi-agent systems: a preliminary report". In Proceedings of the 4th European Workshop on Modeling Autonomous Agents in a Multi-Agent World, Rome.

[12] Y.T. Tang, P. Winoto & X. Niu, 2002. "Investigating Trust between Users and Agents in a Multi agent Portfolio Management System: a Preliminary Report". In the Workshop on business agents and the semantic web, Fifteenth Canadian Conference on Artificial Intelligence.

Indexing Agent for Data Gathering in an e-Travel System

Marcin Paprzycki, Austin Gilbert, Andy Nauli, Minor Gordon, Steve Williams and Jimmy Wright
Computer Science Department
Oklahoma State University
Tulsa, OK 74106, USA
{marcin, austirg, nauli, minorg, stw, jimmyww}@cs.okstate.edu

Keywords: content management, Internet, software agents, data indexing, ebXML registry/repository

Received: October 12, 2003

In this paper we discuss the problem of indexing information available on the Internet with the ultimate goal of delivering personalized content to users of an Internet-based travel support system. We introduce the form of index tokens that will be stored in the system and describe an agent-based subsystem designed to support the indexing function. Finally, we discuss the search agent that was developed to provide the system with index tokens and fueled experimentation with the proposed design.

1 Introduction

In the past decade the travel services market has developed a hugely diverse presence on the Internet, in terms of both resources offered (hotel rooms, rental cars, dinner reservations, golf tee times, "general tourist information," etc.) and approaches to offering them (e.g. aggregation, personalization, mobile delivery). For instance, a simple search using the keyword *hotel* on Google search engine returns about 82,500,000 hits sorted by their rank. Thus, as in most other domains, the potential travel services user must often deal with one of the crucial problems inherent in information diversity: the lack of an encompassing catalogue through which the content **of interest** may be located. Most, if not all, Internet search engines provide only a non-categorized and mostly non-intuitive means of locating and representing data. Furthermore, search results in the travel domain (as well as any other domain) are likely to include too many hits unrelated to actual travel choices. The Google and Yahoo directories are representative attempts to organize access to, and presentation of, many types of data including travel data, however, for instance, they provide no organized booking interface for the data they offer. Additionally, they do not provide any realistic means of personalization of content delivery. Finally, the Google directory consists of a mixture of travel resource types and geographical categories (see [24] for more details) that does not necessarily constitute the best way of supporting travelers. On the other hand, some of the major travel sites such as Expedia, Travelzoo, etc. organize and attempt to personalize a limited subset of travel data (typically airline, car, hotel reservation as well as cruise and vacation package arrangements), based on a limited number of large providers and content stored in tailor-made databases within the system. Here, the mass of information stored on independent Internet sites is completely ignored. Thus, we believe that neither search engines nor the large travel sites are currently capable of providing a complete support to a modern day traveler.

Ideally, a travel support system should act as a filtering and organizing intermediary between travel consumers and travel suppliers [6]. Its primary function [2] is to find the travel information that is most relevant to a given customer at a given moment and deliver it in a well-organized and intuitive way [5, 7, 15, 25]. In order to support this content-delivery role, the system must explore the Internet and other sources dynamically constructing and managing a supply of travel content from known and previously unknown providers [2, 8, 23, 24, 26].

In exploring the potential of such a travel support system, we have followed a two-pronged approach. First, since travel support is a paradigmatic example of the application of agent technology [19, 20], we have decided to utilize software agents as the framework of our system [2]. Second, as an information broker between travel content suppliers and end users (travelers) we must carefully consider the means by which we will structure the information within the system, in order to deliver the most relevant and accurate travel choices to the consumer [4, 5, 29]. We believe that one of the more promising approaches to structuring information from diverse sources is to apply index-based techniques similar to those described in [13] (with references available there). This approach should allow us to effectively deal with data available from multiple sources across the Internet in such a way that pertinent information may be efficiently and accurately selected and delivered to consumers. Note that in our work we are primarily interested in personalized delivery of travel related informational content rather than booking of travel arrangements.

The aim of our paper is twofold. First, we describe an indexing method for storing the travel content. Second we present an agent subsystem that is devoted to management of index tokens in the central repository.

Finally, we briefly describe a simple search agent that has been developed to search the Internet for the travel content and to deliver index tokens to the system.

2 Content management problem

In order for an e-travel support system to accurately reflect available travel options and information, a robust strategy for obtaining this content from sources on the Internet and managing it within the system is required. Existing content provision systems typically approach this problem in one of two ways:

- *by aggregation*: retrieving beforehand **all** information that the system will possibly need in the future, and organizing it in databases in a predefined (by humans) format for future retrieval,

- *by selection*: indexing information to maintain a "map" as to what information (and where) is available on the Internet, and retrieving the actual content only as it becomes necessary to satisfy user's queries.

Most online travel content gateways (e.g. Expedia, Travelersadvantage, etc.) employ the first method, storing the majority of browsable content locally and calling out to the primary source systems on the Internet (e.g. those run by travel providers such as airlines) for verification of locally-cached information (e.g. verified flight schedules, seat availability and ticket prices). The main advantage of this approach is the immediate local availability of content; interestingly, this is also a disadvantage, in that it leads to the problem of "data coherency." In addition, the amount of data that has to be necessarily stored locally and continuous local processing necessary for aggregation systems to operate makes them extremely resource intensive.

The majority of search engines (e.g. Yahoo, Interia, Lycos, etc.) take a hybrid approach, aggregating only a limited store of data (such as page headers and a few selected / cashed pages) necessary to support the search function. This approach attempts at striking a balance between the amount of content stored locally, frequency of local information updates and the precision of the search function. Rudimentary content organization and differentiation available in browsers combined with the relative freshness of data are a reasonable means for satisfying typical content searches (i.e. where the content changes infrequently); however, this approach is wanton when applied to travel-oriented services where the freshness of content is of paramount importance.

Our e-travel system fully embraces the second approach to content management (*by selection*) by attempting to develop a well-organized and highly cross-referenced index of Internet-based content (for a description of a number of similar systems see [1, 10]). The proposed system dynamically utilizes remote content by referencing local indices – pointers. It focuses on the classification of content instead of the content itself, as in a library catalog (or in yellow pages), only storing enough information in indices to satisfy user queries. This approach eliminates the above mentioned problem of data coherency and is aimed at reducing the overall amount of data stored and managed locally. The downside of this approach is that the actual content must always be retrieved from a remote site. If a content provider becomes unreachable, the e-travel system is unable to retrieve the information and thus fails to fulfill the user's request. We have faced such a situation during our system's development process when a remote site providing reverse geo-coding for full addresses ceased to offer geo-coding services, leaving us with a much less desirable choice for our GIS subsystem (see section 5.1). More generally, any latency in communication with the primary content provider is reflected in the performance of the system. Nevertheless, in designing the e-travel system we felt that the advantages of accurate indexing combined with ability to deliver up-to-the-minute information and possible optimization of local resource utilization (resources can be utilized to provide user with personalized content rather than to manage copious volumes of data) outweighed the disadvantages of remotely-stored content. We also expect that an approach based on indexing will improve the limited queries options and result displays caused by traditional database logic and principles [1, 13].

3 Agent-based travel support system

3.1 History

The initial design of the travel support system was presented in [2, 23, 26] and while it is being constantly modified (this paper represents such a modification), the general idea of dividing the functionality into two coordinated subsystems, one handling content management and the other content delivery [6], remains unchanged. In this paper we concentrate on the content management aspects of the system. Further details about the systems and in particular the proposed content delivery functionalities can be found in [2, 7, 8, 15, 24]. The initial development of the system was initiated using the Grasshopper agent platform [9], however quickly realized that it did not fully supported the FIPA [5] standards at that stage of its development. We have therefore switched to the JADE agent platform [14], which is built around the FIPA standards.

3.2 Sources of content indices

The travel options and information that is presented by the e-travel system originates from two types of sources on the Internet: verified and unverified. Verified sources are referred to as *Verified Content Providers* (*VCP*). This designation implies a degree of conformance to expected standards of accuracy, format, and availability of described travel options. Content from *VCP*s can be either fed directly to the system or gathered by search

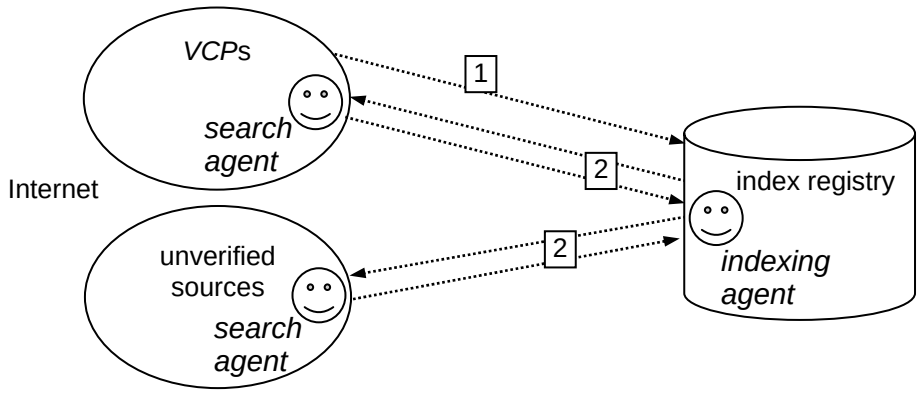


Figure 1: Information gathering and indexing; 1 – flow of index tokens originating from the *VCP*s, ready for insertion to the registry, 2 – flow of index tokens resulting from the Internet searches

agents, as described in [2]. In the first case we assume that the incoming *index tokens* (pointers to available information) are both in the required format and complete, and thus can be immediately stored in the system without further processing. In the second case, the acquired content indices may be incomplete and/or require further processing. When dealing with unverified sources the situation is similar to the latter case with an added component of necessary verification and deconfliction of remote information. At this stage of system design we will omit these last two issues of verification and deconfliction, assuming they have been successfully resolved. Let us note, that the proposed approach allows us to address one of the important research issues raised by Nwana and Ndumu in [20]; how to deal with dynamically changing content and form of the Internet-based information. Here, we assume that the *VCP*s are in contractual agreement with the travel agency and either they will continue to deliver index tokens in prescribed format, or any changes in their site design will be communicated to our system, allowing it to be adjusted accordingly. Since the *VCP*s are the primary sources of the information, changes occurring in the remaining sites do not threaten the functioning of our system. Furthermore, this approach allows us to avoid most questions related to the reliability of Internet-based information. Finally, since the *VCP* provided information is assumed to be trustworthy, we can rely on them as the source of accurate information delivered to the user, while other, unverified, sources can be utilized only as supplementary resources. Regardless of source, the acquired indices are stored in the central registry for later access by the content delivery functions of the system. When the user requests information, a relevant content pointer is either found in the registry and the process of content extraction from the provider(s) is initiated (while additional search agents may be released to the Internet seeking additional content relevant to the query; in order to focus our presentation we will omit discussing this possibility), or a new index search and acquisition is forced in order discover relevant content (from both *VCP*s and unverified sources). Since the case of complete tokens being delivered directly by the *VCP*s is trivial (only an indexing agent is required to receive them and correctly store in the system), for the remaining part of this paper we will concentrate our attention on the tokens resulting from the Internet searches.

3.3 Semantics

Ideally, the content management subsystem should shield the rest of the e-travel system from the supply / retrieval mechanics of the travel content. Additionally, it should allow the content delivery functions of the systems to operate on the assumption that travel information is accurately classified. In theory, this would require the content management subsystem to semantically "understand" the information it keeps track of [8, 15, 25]. Here we have to acknowledge that currently available technology does not support this assumption of semantic "understanding" (its foundations are being developed, but are not widely accepted and thus cannot be assumed). In the absence of such technology, our system attempts the next best substitute. We apply a predefined categorical overlay to the travel information managed by the system, and allow the entire system to tune the accuracy of this overlay (e.g. with user, agent and supplier feedback, as described in [6]), with the ostensible goal of simulating real semantic classification. In addition, we pay close attention to the efforts initiated by the Open Travel Alliance that attempts at introducing a hierarchical description of the "world of travel" and most important processes taking place there [22] (see also [24] for more details). Note that, while currently not operating on the semantic level, most of the functions of the proposed system can be adjusted to involve, for instance, RDF / OWL based ontology / semantics [11].

4 Structure of index tokens

The e-travel system relies heavily on the accuracy and completeness of local content indices. They must be succinct enough to be easily acquired and stored, yet verbose enough to satisfy all of the requirements of both content management and content delivery subsystems. Consider the following scenario: a user wishes to make travel arrangements to visit Mt. Rushmore, a historical monument. The user must first travel to South Dakota (requiring a means of transportation), and perhaps find a place to stay (hotels in the area). She may also wish to know about local restaurants or other places of interest. In order to satisfy the user's request for travel arrangements, the system must initially make two major distinctions based upon the query alone: location (South Dakota) and desired destination/attraction (Mt. Rushmore). In addition, the e-travel system must also be able to resolve multiple providers of content relating to Mt. Rushmore, in order to find those indices, which will eventually yield the most desirable response for the user (for the purpose of this paper we skip the question of content provider ranking, which is one of the possible ways of dealing with potential information overload).

Our current design of indices evolved from our early attempts to develop a classification system of the world of travel content [2], and was adapted to satisfy the above requirements. We now describe an index as a tuple consisting of:

(<*provider*>,<*type*>,<*location*>,<*?notes?*>)

Here, the *?notes?* component is added to the tuple to support of various administrative functions necessary when dealing with data delivered by the search agents (for more details see Section 5). Let us now look into the *provider, type* and *location* fields of the tuple in more detail.

4.1 The *provider* component

The *provider* component describes the means of accessing travel resources on the Internet. It is stored in the form of a Uniform Resource Identifier (URI). This URI describes the access method for the resource, the location of the resource, and any marker data that may be unique to this resource within the provider. In addition to explicitly identifying the transport protocol, the protocol section also (directly or indirectly) identifies the access methods of the server. For example, http:// and ota:// each have their respective access methods (hypertext and Open Travel Alliance protocols). Other possible protocols include edi:// and soap://. The URI also contains the host name to communicate with using this protocol. Let us also note that our system is capable of efficiently dealing with situation when multiple providers supply information pertinent to a given travel resource. In this case multiple index tokens varying only in the *provider* component will be "co-stored" in the repository for efficient retrieval (for more details see [30]).

4.2 The *type* component

The *type* component of a tuple describes the position of a travel resource in the taxonomic hierarchy of all resources (e.g. Accommodations -> Hotels -> Chains). The system will utilize this information to filter out content that for some reasons (i.e. in the context of a given query, or for a particular user) is not pertinent to a user's needs. Thus, it is the focal point for the proto-semantic division of travel information. For example: if the user is interested in hotels, an agent will be able to retrieve only hotel indices from the repository. Current version of our hierarchical taxonomy for the *type* component is derived from the modified Yahoo! directory of Travel and the Open Travel Alliance [22] XML Schemas (see also [24] for more details). The content type is intended to define the relationships between travel resources.

4.3 The *location* component

Geography and location are key factors for determining the relevance of indexed travel resources to a particular user's travel plans. The *location* component must be flexible enough to support the multiple ways it may be utilized. Location information must be specific enough to differentiate between different sites. It must be hierarchical so that organizational relationships between sites at different locations on different levels (continent, country, state, city, et al.) can be surmised (e.g. the destination is in a different country). Given these criteria, our initial design of the *location* component consists of: a taxonomic description based on the ISO-3166 standard, which defines the continent, country, state or province, and city; and the latitude and longitude for exact locations and proximity searches. These are represented in the *ebXML* hierarchy [21]. However, in the tuple itself we store the geographical information in the form of a (latitude, longitude) pair. This form as been selected due to the need of processing geospatial information beyond simple information that a given place of interest is, for instance, located in Claremore, Oklahoma, United States.

The *type*/*location*/*provider* tuple as described above, located in hierarchical structures representing resources and geospatial locations, is the basis of the classification scheme to be utilized by all of the functions of the travel support system, from the retrieval of content from travel suppliers on the Internet to the delivery of travel choices to the end user. It is with these functions in mind that we proceed to manifest the tuple on the implementation level, and, we hope, provide an efficient means of communicating travel content. Let us also observe that the proposed schematic solves the, above indicated, problem of the Google directory [24]. In our approach we are able to untangle the geospatial information from the travel resource information by providing two separate but complimentary "looks" at our data. In this way, we are also making an initial step toward developing ontology of travel.

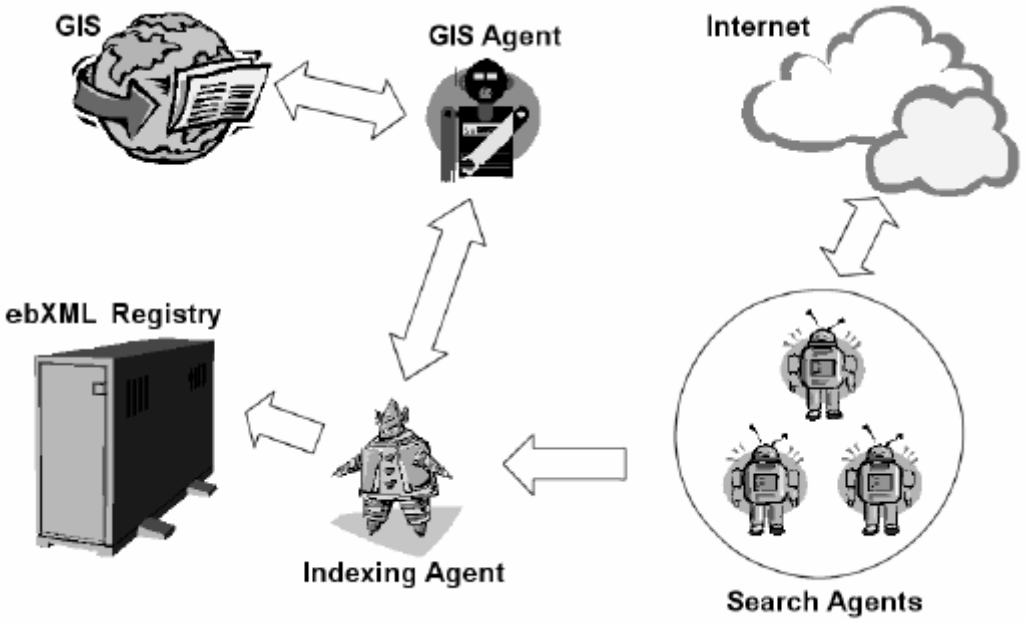


Figure 2: Proposed architecture for indexing travel data from the Internet.

The following is an example of a complete index token that is ready to be stored in the system (the *?notes?* filed is omitted, but in this case it would contain information that the token is complete and no further processing is required):

(edi://www.drp_sushi_palace.com/, restaurant, (25'45'', 34'67''))

Here, information about a *restaurant* is available at *www.drp_sushi_palace.com* and the communication protocol with that site is *edi://* and the location of the restaurant is *25'45'', 34'67''* (the details of the ISO-3166 location will be retrievable from the position of the token in the geo-tree structure in the registry, while the restaurant is positioned within a hierarchical structure of types). Once a complete index token is successfully inserted into the registry, it is ready for processing by the content delivery subsystem (as described in [2, 7, 8, 15, 24]) and can be utilized to prepare responses to user queries. To implement the storage of index tokens, we have decided to utilize the turned to the ebXML Registry / Repository (for an extended discussion of index storage see [21, 30]).

5 Index acquisition

We now consider the actual process of index acquisition. As indicated above, there are two separable sources of index tokens: *VCP*s that feed complete indices directly to the system (this relationship is pre-defined by agreements between selected providers and the e-travel system); and *search agents*, which explore both the remaining *VCP*s and other repositories on the Internet. Tokens acquired by *search agents* may or may not be complete, and if their source is unverified, the content referred to should be validated and deconflicted (in the case when there is no way to verify the information, the *?notes?* field will be utilized to store such an information so that in the content delivery subsystem such information can be treated accordingly, when delivered to the user). Within the system all incoming tokens (from the *VCP*s and *search agents*) are received and handled by an *indexing agent*, which inserts them into the *registry*. Incomplete or not yet validated tokens are marked as such in the *?notes?* field of the index tuple. Furthermore, incoming tokens may have various priority levels, also indicated in the *?notes?* field. For instance, tokens acquired by the *search agents* for a user currently interacting with the system will have to be made ready for use (completed, and if necessary validated and deconflicted) as quickly as possible, while other tokens (it is assumed that in a fully operational system *search agents* continually traverse the Internet in search of travel-related information, similarly, for instance to the Google-bots) may be processed when the system is "idle." Thus the content management subsystem, as we described it so far, consists of index tokens being fed directly by the *VCP*s and *search agents* that find location of pertinent travel related content and generate index tokens; furthermore we have one or more *indexing agents* that store index tokens in the registry (number of *indexing agents* will depend on the scalability needs of the system). The JADE-based implementation of our system helps facilitating agent interactions. First, communication between the *search agents* and the *indexing agent(s)* (as well as all other inter-agent communication) is facilitated using ACL messages which are implemented in compliance of the FIPA standard. Second, *search agents* can locate *indexing agent* by simply querying the JADE Directory Facilitator. The following code snippet illustrates the method invoked to achieve this goal:

```
AID index = new AID();
DFAgentDescription dfd = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("IndexingAgent");
dfd.addServices(sd);
try {
  while (true) {
    SearchConstraints c = new SearchConstraints();
    c.setMaxDepth(new Long(3));
    DFAgentDescription[] result =
    DFService.search(this,dfd,c);
    if ((result != null) && (result.length > 0)) {
      dfd = result[0];
      index = dfd.getName();
      System.out.println("Found Indexing Agent");
      break;
    }
  }
}
catch (FIPAException e) {
  System.out.println("Cannot Find Indexing Agent");
}
```

When this method returns, the variable index will be pointing to the Agent Identifier of the *indexing agent*.

5.1 The *GIS agent*

As noted above, the location component of the index token tuple is to be represented as a (latitude, longitude) pair. Typically, geospatial data available on the Internet is not represented in such form. Thus most index tokens delivered by the *search agents* will not have the correct form; typically an empty location field and a note specifying token's incompleteness in the *?notes?* field. To deal with this situation as well as to support a number of other important functions in the content delivery subsystem a *GIS agent* has been developed. In the context of this paper, the main role of the *GIS agent* is to fill the (latitude, longitude) data of the index token. This is the standard reverse geo-coding function, where the input is an address (found within a web resource by the *search agents*) and output is the (latitude, longitude) pair. Current implementation of the *GIS agent* relies on external party to provide reverse geo-coding functionality. As mentioned above, initially we were able to locate a service which provided the required functionality free of charge, however, shortly after completing the initial implementation of the GIS-agent, this service was discontinued. Thus, in our subsequent experiments we have utilized the http://mapper.acme.com site. This website accepts the request for GIS queries interactively (e.g. using a form and an input box). This can be easily transformed using Java's HttpURLConnection class. The form is submitted using the get method and thus it can be represented as a URL by appending the base address of the website with parameter and value pair of intended queries, e.g.

http://mapper.acme.com/find.cgi?zip=74075.

While, obviously, this particular service is only of limited capability – it accepts only ZIP codes and only of locations in US – this level of detail provided by a free of charge system was satisfactory for our proof of concept system. Obviously, in a real system geospatial information would have to be more precise than one that is based solely on ZIP codes. Such information is available (including locations outside of United States) and can be easily incorporated into our system. Unfortunately, services delivering robust reverse geo-coding are not freely available and we have decided to continue utilizing the ZIP-code only service for the time being.

Summarizing, in the current implementation of the content management subsystem, the *GIS agent* receives the ZIP code information from the *indexing agent* (send as an ACL message) and contacts the acme.mapper.com site to obtain the (latitude, longitude) pair (our implementation utilizes a slight shortcut as the search agents deliver also the ZIP code instead of a token with an empty location field; this latter solution that was postulated above requires implementation of auxiliary agents; see Sections 5.2). The resulting information is send back to the *indexing agent* (again, as an ACL message) which then completes the token and inserts it in the *repository*. Overall, the simplified schema of the system has been depicted in Figure 2.

As noted earlier, in a travel support system, there is a need for a much broader support for geospatial data processing. For instance, it will be necessary to respond to distance oriented queries, such as "how far is it from a given restaurant X to a given movie theater Y," or "which restaurants are within a given distance of hotel Z." These functions can either be implemented inside of the same *GIS agent* or each of the particular sub-functions can be implemented as a separate *GIS agent*. While the second solution seems to follow more closely the spirit of agent system development (where separate functions are represented by separate agents), and the particular GIS functions are naturally separated by the content management and delivery subsystems, the final decision for the agent-based implementation of the geospatial functions will be made in the next iteration of system development.

5.2 Auxiliary agents

As discussed above, one of the problems in indexing data originating from the Internet is the need for dealing with incomplete index tokens returned by the search agents. No agent can acquire information that is simply not available. As indicated above, the majority of content providers do not provide geospatial information in the form desired by our system. Rather they feature an address (complete or partial). Thus the system will have to properly manage incomplete index tokens (at this stage we will consider any token gathered from an unverified source as incomplete). To achieve this goal, incomplete tokens are flagged as incomplete in the *?notes?* field, assigned priority and inserted into the registry. They are then processed by token completion, validation, deconfliction (*CVD*) *agents*. These agents traverse the registry and process the incomplete or unverified tokens. As an example let us consider the case of a token that is missing the location data. It is known who is the *provider* of the data, the *type* is also known (it is a hotel), while the *location* field contains no data and

the *?notes?* field specifies an incomplete token with high priority. The *CVD agent* will therefore create an instance of a *query agent*. This agent will communicate with the content provider (using the specified protocol available from the *provider* field) and establish that the hotel in question is the Fairmont Hotel in San Francisco and recover its street address (from the provider, or from a different content provider discovered during separate web-searches). This information will be returned as an ACL message to the *CVD agent* responsible for managing this particular token. The *CVD agent* will then contact (via an ACL message) the above described *GIS agent* (see [2, 23, 26] for more details) where reverse geo-coding will result in the (latitude, longitude) pair. This information will be then inserted it into the token and the flag signifying an incomplete token will be removed from the *?notes?* field, thus making it a full member of the *registry*. While, currently, this functionality is not yet implemented, its implementation is one of our next goals in the development of the system.

6 Content gathering

The difficult problems of content indexing and retrieval are representative of a crucial issue confronted in Internet-related research: how to introduce "understanding" to machine-web interaction. One of the reasons that many online content gateways choose the aggregation approach to content management is because it is easier to implement, despite its resource-intensiveness. The more "intelligent", selective approach of indexing content for later utility requires an in-depth, machine "understanding" of the content in order to reliably utilize it.

6.1 Interpreting sources

In recent years there has been a resurgence of interest in ontologies as a way of dealing with the problem of machines "understanding" the semantics of information on the web. Many claim that agents with ontologies will be the next breakthrough technologies for web applications [11]. This has been the thrust of the Semantic Web project [27] – the development of an ontology-described content infrastructure that will allow machines to interpret semantics as opposed to mere syntax. This capability has been realized in web pages hosted by several organizations. According to the DAML Crawler [4], as of the time of our writing, there are semantically 21,025 annotated web pages. Unfortunately, this number is negligible compared to the total of 7 billion web pages on the Internet. Therefore, today, it is not realistic to assume that agents can simply understand the web-content.

The design of our e-travel system takes into account the eventual existence of a semantically-described web; and, in particular, development of a complete and generally accepted ontology of travel, but it does not rely on it. Rather, we plan to implement an intermediate solution that allows us to depend on agent "understanding" only within the e-travel system, a working assumption which is supported by adapting the perimeter of the system (i.e. the index acquisition system) to simulate semantic gathering [11, 27].

One of the typical approaches to developing agents with the necessary functionalities is through topical web crawlers [16]. Topical web crawlers take advantage of knowing the context of the query to differentiate between the relevant and irrelevant web pages. Web pages are considered to be relevant if their similarity value satisfy a given threshold. Similarity value is calculated based on lexical analysis of the web page.

Another approach to semantic understanding of the web is through application of wrappers. For example, information agents in Heracles [13] are trained to locate meaningful information in the web pages by being shown examples consisting of web pages labeled with markers to indicate where the information is located. These examples are then used to develop a set of wrappers that are subsequently utilized in intelligent searches.

6.2 Simple search agent

While acknowledging that the above described techniques are already relatively sophisticated and new techniques are constantly being developed, for the purpose of our demonstrator system we have decided to pursue a more simplistic approach. Note, however, that we rely here on one of the important advantages of agent-based system design. Our *search agents* were implemented to verify the design of the system, to fill-in the *registry* with tokens, to pursue initial efficiency and scalability studies. As the system matures, our simple *search agents* will be replaced by more sophisticated *agents* and the system will continue its work without any additional changes.

The *search agent* must be designed so that it can classify a web page into the correct travel resource and finding necessary information to create its index token. Our approach is to use a simple statistical method to calculate the similarity of a web page to a set of given keywords (or query). This statistical method compares the content of the web page with keywords that represent a travel resource. The similarity value of the web page and the keywords are then computed. This value is then used to decide if the given web resource matches the travel resource. In implementing this functionality we have utilized existing software.

In designing our *search agent*, we utilized several software packages. Assume that our agent accessed a web page. First, the HTML Parser [12] was used to strip out all HTML-based formatting instructions. The stripped-out HTML page was then fed to the Apache Lucene [3] for statistical analysis. The statistical analysis process begins with applying a lower case filter, which turns all words into lower case. The second step consists

of removing the stop-words (words with no meaning e.g. *the*, *than*, *of*, *which*, *were*, *are*, etc.). This allows us to reduce the size of the index file. The list of stop words was based on [28]. In the third step, Porter Stem Filter [19] was applied to convert words into their basic form (e.g. *running* into *run*, *watches* into *watch*). The final step was to compute the statistical similarity of the filtered content to the travel resource keywords provided to the system. The Apache Lucene package includes all these steps. It also implements the vector space model to calculate the similarity value. The vector space model works by comparing the frequency of words that appear in the document with a set of given keywords using the formula:

$$similarity(d_j, d_k) = \frac{\sum_{i=1}^{n}(td_{ij} \times tq_{ik})}{\sqrt{\sum_{i=1}^{n} td_{ij}^2 \times \sum_{i=1}^{n} tq_{ik}^2}}$$

where td_{ij} denoted the i^{th} term in the vector for the document j, tq_{ik} denotes the i^{th} term in the query vector k and n number of unique terms in the data set.

We have experimented with the above described simple search agent in two ways. First, to obtain some indication of the approach's accuracy, for this purpose we have implemented a GUI front end to communicate with JADE agents. As expected, the simplicity of the approach yielded disappointing results, with correct identification only about 25% of the visited web pages when a single keyword was used. At the same time these results seem promising, as much better recognition rates are obtainable (more details about our experiments and their results can be found in [18]). More importantly, we were able to develop a working system in which *search agents* scavenged the web and produced index tokens. These index tokes were completed through interactions between the *indexing agent* and the *GIS agent*. Finally, the *indexing agent* was able to utilize the Java API for XML Registry (*JAXR*) to insert completed tokens into the ebXML Reiztry /Repositiry.

7 Concluding remarks

In this paper we have reported on our progress in developing an agent-based travel support system. Our principal motivation for attempting to implement a realistic agent system was to establish the potential and limitations of a more general class of agent-based systems. In this we follow the methodological lead of Nwana and Ndumu [20] who have stressed the importance of the implementation and experimentation phases of agent system development. We are also challenged by the fact that all the past projects have been limited in scope [13, 19, 25, 29] or abandoned in early stages of development.

At the time of writing of this paper we have implemented (1) the hierarchical classification schemes for the *type* and the *location* components and instantiated them in the *ebXML registry / repository* [21, 30]; (2) the *simple search agent*, the *indexing agent* and the *GIS agent*; (3) communication between them. In this way we were able to perform initial experiments with inserting and storing tokens in the *registry*.

These initial experiments indicate that we will have to rethink the way in which the index tokens are stored and operated on. For still unknown reasons we have run into a number of problems with the ebXML Registry/Repository. While all necessary operations worked well when its native GUI interface was used, we were constantly running into problems when combining the Repository with the JAXR and other insertion techniques. Some of these problems were of technical nature e.g. hanging registry, runaway threads etc., but many were also related to scalability e.g. attempting to instantiate the complete ISO-3166 classification for the United States in the Repository proved impossible. Since it seems unlikely that the Registry/Repository scalability issues were related to hardware shortcomings (a 2.4 Ghz Pentium-4 server with 1 Gbyte of RAM was used), we tend to believe that this may be a problem with the currently existing *ebXML* implementation. Establishing this fact was one of the important lessons learned from our experiments. This will force us to re-evaluate the token storage technology before the next step in system design and implementation. Observe, however, that while the token storage technology may change, this will not affect other parts of the system.

Obviously, we recognize the drawbacks of relying on third-party GIS subsystem as the primary source for latitude and longitude information (limited to US and Canadian address only). However, as indicated above, we consider such drawbacks to be insignificant during the system development time. Finally, our categorization of the world of travel (the hierarchy used to structure the information stored in the *type* field) is primarily based on the Yahoo! catalog and the work of the Open Travel Alliance (OTA) [22] and, obviously it needs to be re-thought and improved on the basis of our experiments.

This leads us to the obvious fact that there exist a large number of research and/or practical issues that need to be addressed in the near future. Let us list some of them (obviously this is only a partial listing): (1) re-evaluation of the index storage technology, with a strong possibility of replacing the ebXML Registry/Repository by a more robust solution; (2) completion of the content management subsystem as described in this paper (including the auxiliary agents) – this would allow us to launch the system to automatically collect index tokens and populate the registry for further experiments; (3) addressing the question of search agent intelligence – we would like them to be effective in filtering web content and supplying our system with complete index tokens while being relatively lightweight – and, definitely we need high reliability results when we will start to automatically populate the repository (here it will be better to reject a correctly categorized resource than to

accept an incorrectly categorized one); (4) investigating how many agents of various types (indexing, token completion, search, GIS etc.) are required to prevent processing bottlenecks in the content management subsystem; (5) evaluating if the proposed indexing schema is robust enough to support the content delivery functions; (6) study how does the proposed indexing schema match with the personalization oriented functions that the system is to support (in particular user behavior data storing and mining [5]). Our experimental findings (like the fact that the ebXML Registry/Repository may not be capable of supporting our needs) indicate that the above listed research questions will have to be investigated both theoretically and practically. As suggested in [20], experimentation will play the crucial role of guiding our system development. We will report on our progress in subsequent publications.

References

[1] Abramowicz, W., Kalczyński, P., Węcel, K. (2002) *Filtering the Web to Feed Data Warehouses*, Springer Verlag Publishing, New York.

[2] Angryk, R., Galant, G, Gordon, M., Paprzycki M. (2002) Travel Support System – an Agent-Based Framework, *Proceedings of the International Conference on Internet Computing* (IC'02), CSREA Press, Las Vegas, pp. 719-725

[3] Apache Lucene from http://jakarta.apache.org

[4] DAML Crawler, retrieved Feb 11, 2003 from http://www.daml.org/crawler/

[5] FIPA, http://www.fipa.org

[6] Galant V., Jakubczyc J., Paprzycki M. (2002) Infrastructure for E-Commerce, in Nycz M., Owoc M. L. (eds.), *Proceedings of the 10th Conference on Knowledge Extraction from Databases*, Wrocław University of Economics Press, pp. 32-47

[7] Galant V., Paprzycki M. (2002) Information Personalization in an Internet Based Travel Support System, *Proceedings of the BIS'2002 Conference*, Poznań, Poland, pp. 191-202

[8] Gordon M., Paprzycki M., Galant, V. (2002) Knowledge Management in an Internet Travel Support System, in: Wiszniewski B. (ed.), *Proceedings of ECON2002*, ACTEN, Wejcherowo, 2002, pp. 97-104

[9] Grasshopper, http://www.grasshopper.de

[10] Gudivada V, Raghavan V, Grosky W, Kasanagottu R, (1997) Information Retrieval on the World Wide Web, *IEEE Internet Computing*, pp. 57-68.

[11] Hendler, J. (2001) Agents and semantic web, *IEEE Intelligent Systems Journal*, 16(2), pp. 30-37

[12] HTMLParser, http://htmlparser.sourceforge.net

[13] Intelligent Systems for Tourism (2002) *IEEE Intelligent Systems,* pp. 53-55

[14] JADE, http://jade.cselt.it

[15] Jakubczyc, J., Galant, V., Paprzycki, M., Gordon, M. (2002) Knowledge Management in an E-commerce System, *Proceedings of the Fifth International Conference on Electronic Commerce Research*, Montreal, Canada, CD, 15 pages

[16] Menczer, F., Pant, G., and Srinivasan, P. (2003) Topical Web Crawlers: Evaluating Adaptive Algorithms, *ACM Transaction on Internet Technology*, 5(N), pp. 1-38

[17] Porter M., Porter stemming algorithm, http://www.tartarus.org/~martin/index.html.

[18] Nauli A. (2003) *Using software agents to index data for an e-travel system*, Masters Thesis, Oklahoma State University, 2003

[19] Ndumu, D., Collins, J., Nwana, H. (1998) Towards Desktop Personal Travel Agents, *BT Technological Journal*, 16 (3), pp. 69-78

[20] H. Nwana, D. Ndumu, A (1999) Perspective on Software Agents Research, *The Knowledge Engineering Review*, 14 (2), pp. 1-18

[21] OASIS/ebXML Registry Services Specification v2.0, http://www.oasis-open.org/ committees/regrep/ documents/2.0/specs/ebrs.pdf

[22] Open Travel Alliance, http://www.opentravel.org

[23] Paprzycki M., Angryk R., Kołodziej K., Fiedorowicz I., Cobb M., Ali D. and Rahimi S. (2001) "Development of a Travel Support System Based on Intelligent Agent Technology," in: Niwiński S. (ed.), *Proceedings of the PIONIER 2001 Conference*, Technical University of Poznań Press, Poznań, Poland, pp. 243-255

[24] Paprzycki M., Gilbert A., Gordon M., Wright J. (2003) The World of Travel: a Comparative Analysis of Classification Methods, *Annales UMCS Informatica*, A1, pp. 259-270

[25] Paprzycki M., Gordon M., Gilbert A. (2002) Knowledge Representation in the Agent-Based Travel Support System, in: Yakhno T. (ed.), *Advances in Information Systems, Springer-Verlag*, Berlin, pp. 232-241

[26] Paprzycki M., Kalczyński P. J., Fiedorowicz I., Abramowicz W. and Cobb M. (2001) "Personalized Traveler Information System," in: Kubiak B. F. and Korowicki A. (eds.), *Proceedings of the 5th International Conference Human-Computer Interaction*, Akwila Press, Gdańsk, Poland, pp. 445-456

[27] Semantic Web, http://www.semanticweb.org

[28] Stop Words List, http://www.onjava.com/onjava/2003/01/15/examples/EnglishStopWords.txt.

[29] Suarez J. N., O'Sullivan D., Brouchoud H., Cros P. (1999) Personal Travel Market: Real-Life Application of the FIPA Standards, *Technical Report*, BT, Project AC317

[30] Wright, J., Williams, S., Paprzycki, M., Harrington, P. (2003) Using ebXML Registry/Repository to Manage Information in an Internet Travel Support System, in: Abramowicz W. and Klein G. (eds.), *Proceedings of the BIS'2003 Conference*, Poznań University of Economics Press, Poznań, Poland, 2003, pp. 81-89

Multi-Agent System Case Studies in Command and Control, Information Fusion and Data Management

Frederick Sheldon and Thomas Potok
Oak Ridge National Laboratory
Computational Sciences and Engineering
Oak Ridge, Tennessee 37831-6363
Phone: 865-576-1339/ Fax: 865-241-6275
Email: SheldonFT@ornl.gov | PotokTE@ornl.gov

Krishna Kavi
Department of Computer Science
The University of North Texas
Denton, Texas 76203
Phone: 940-565-2767 / Fax; 940-565-2799
Email: Kavi@cs.unt.edu

Keywords: Intelligent Software Agents, Ontology, Information Fusion, Collaborative Decision Support

Received: August 18, 2003

On the basis of three different agent-based development projects (one feasibility study, one prototype, one fully fielded), we assess the fitness of software (SW) agent-based systems (ABS) in various application settings: (1) distributed command and control (DCC) in fault-tolerant, safety-critical responsive decision networks, (2) agents discovering knowledge an open and changing environment, and (3) light weight distributed data management (DM) for analyzing massive scientific data sets. We characterize the fundamental commonalities and benefits of ABSs in light of our experiences in deploying the different applications. [1]

1 Introduction

Systems whose information-processing structures are fully programmed are difficult to design/evolve for all but the simplest kinds of applications. Changing and dynamic open environments will characterize future real-world software application context. Such systems must be able to modify their behavior by changing their information-procession structures [1]. Software agents (SAs) are the latest advancement in the trend toward small modular pieces of code where each module performs a well-defined, focused task or set of tasks. Programmed to interact with and provide services to other agents, including humans, SAs autonomously with prescribed backgrounds, beliefs and operations. Systems of agents can access and manipulate heterogeneous data such as information available on the Internet [2]. Not all agent systems have to have the above properties but any agent-based paradigm must have the ability to engender agents with some or all of the aforementioned properties.

1.1 Agent Technology, Maturation & Evolution

SW development methods have been transformed over the years from structured analysis methods, where processing and data were kept separate [3], to Object-oriented (OO) methods, where processing and data are combined into SW entities called objects [4, 5] (¶1.6-1.7). Object technology was further enhanced with distributed capabilities, allowing an object on one system to communicate with objects on other systems [6]. Objects may be transmitted across a trusted network and executed on another computer, commonly known as mobile code [7].

Furthermore, component-based software development (CBSD) can be viewed as a similar evolutionary trend, which differs from traditional software development. For example, CBSD includes activities selection and creation of SW architectures, as well as the customization of components, while implementation deals with component integration. Typically, this process involves developing wrappers that bond reusable components into a cohesive system rather than extensive coding "from scratch" construction. Indeed, developers must architect/design extensibility into a system and all of its parts to make

components independently producible and deployable. SAs offer a great deal of flexibility and adaptability within this context. Agent-oriented SE provides developer's high-level flexible abstractions from which to represent and conceptualize distributed application systems (e.g., delegation of information search, analysis, negotiation and presentation).

SA systems, to some degree, are characterized by being persistent, mobile, knowledgeable, adaptable, autonomous and collaborative, which facilitates the building and evolving of software systems as technologies and requirements change [8]. Developers use increasingly pervasive message-based middleware and component technologies, Java, Extensible Markup Language, and the Hypertext Transfer Protocol to create agent-based software systems. Mobile appliance-oriented application servers and portal technologies based on these technologies provide a basis for more robust agent-oriented systems. These technologies will make the use of mobile appliances, adaptive content, and SAs quicker and easier.

1.2 Distributed Computing

Distributed or ubiquitous computing envisions devices ranging from super computers to nanoscale CPUs acting in concert to solve problems. Current distributed computing approaches include the Common Object Request Broker Architecture (CORBA), the Distributed Component Object Model (DCOM), and Remote Method Invocation (RMI) Each provides a way of executing a SW function needed by one computer on a different computer. Remote execution places a number of constraints on the SW. For example, assume that a source object (e.g., program or function) is attempting to execute some function on a target object; the source object must have the capability to resolve the network and computer memory address of the target object. Next, the source object must have detailed prior knowledge of the functions (methods) and parameters available on the target object, as well as return information. There are also assumptions that these remote functions will be accessed synchronously and that the network connections are available and permanent. If any of these assumptions does not hold, then these distributed interactions will fail [9].

1.3 Agent infrastructure

The dynamic interaction of multiple SAs requires an architecture that supports "our definition" of an agent (i.e., is a program P, written in a language l, P_l an agent?), what underlying infrastructure is needed to support agents to interact effectively, and how the agents will utilize the infrastructure to interact. The Oak Ridge Mobile Agent Community (ORMAC) is a communication/mobility framework developed over the course of several agent-based research projects. ORMAC is generic framework providing transparent agent communication and mobility across Internet connected hosts (Fig. 1). This architecture enables an agent community to be quickly created using a set of machines with each machine executing the ORMAC agent host software (SW): (1) SAs migrate among machines as necessary to facilitate communication among agents within the community, and (2) ORMAC SAs can also interact with systems and agents that are not part of the community. Internet mobility is very limited based on enforced Internet security/firewall constraints. ORMAC uses the Foundation for Intelligent Physical Agent (FIPA) compliant agent communication language (ACL) messages. Any FIPA compliant agent can interact with an ORMAC agent [10, 11]. Within an ORMAC community, each agent host has a name server responsible for tracking where agents are currently hosted. In addition, the name server is responsible for answering queries from agents trying to locate other agents in the community. For example, an agent may want to broadcast information to all agents within the community. The name server for each agent host is used to locate all such agents for delivery of said message(s).

Agents migrate among machines by changing *agent hosts*. When an agent is received at an agent host, the agent host provides it with an agent context. This agent context is the agent's only point of contact with the machine it is running on and provides machine specific environments for the agent to work. The agent is not allowed to directly communicate with the agent host or other agents. This provides an architectural layer for security in the ORMAC system (written in JAVA, ORMAC uses Remote Method Invocation (RMI) to communicate among agents).

1.4 Heterogeneous agent interoperability

Ontology-based thesauri have been an important part of research in Natural Language Processing. As the need for distributed software configurations has risen, Ontologies have become increasingly important. Ontologies have evolved as a convenient way to permit agents using diverse vocabularies to specify common concepts. There are two

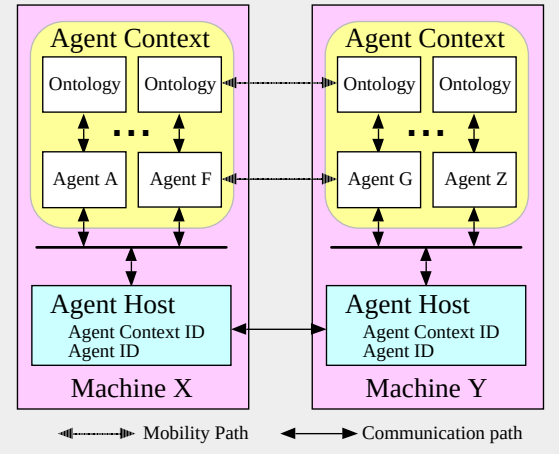


Fig. 1 ORMAC mobility-communication architecture.

main approaches (1) creating a large general ontology, or (2) many domain-specific ontologies. Most ontologies or thesauri are constructed manually, however, methods have been developed for automated construction of such [2]. In our Virtual Information Processing Agent Research (VIPAR) case study, agents use a flexible RDF (Resource Description Framework) ontology to transform heterogeneous HTML documents to XML tagged documents, and their ability to rapidly cluster newspaper articles that arrive in an asynchronous manner.

Agents move from one machine to another by changing *agent hosts*. The *ontologies* move with the agents. When an agent is received at an agent host, the agent host provides it with an agent context. This agent context is the agent's only point of contact with the machine it is running on and provides machine specific environments for the agent to work. The agent is not allowed to directly communicate with the agent host or other agents. This provides an architectural layer for security in the ORMAC system (written in JAVA, ORMAC uses Remote Method Invocation (RMI) to communicate among agents).

1.5 Independent asynchronous communication

Agent-based architectures provide several advantages over OO technologies where objects communicate through messages. The sender object must know the address of the receiver object (i.e., public methods). In contrast, the ORMAC framework imposes a communication protocol that allows messages to be sent without having to know the specific address/method(s) of the recipient [12]. This allows agents to migrate among host and still be in connected with other agents via direct or broadcast requests to any number of other agents. ORMAC provides the ability to use an ontology to direct agents through a task. An ontology can act as a script, or rule base for an agent to follow. This difference is perhaps more conceptual than practical because there currently is very little ontology standardization. For example, in our information fusion case study, an RDF ontology is used to describe the characteristics of each Internet newspaper within the system while agents use the ontology to correctly interpret and retrieve the appropriate information (see ¶2.2).

Furthermore, agents can suspend processing on one machine, move to another, and resume processing. In this way, the possibility exists to prioritize agents (tasks) by sending high priority agents to faster resources, and for example, load balance a system depending on the workload of each agent. The priority and/or allocation of agents can be determined cooperatively thereby preempting the need for global scheduling (to avert single-point-failure risk).

1.6 Agents cognitive development framework

Kavi, et. al. [13, 14] present a framework for modeling, analysis and construction of agent-based systems. The framework is rooted in the Belief Desire Intention (BDI) formalism and extends the Unified Modeling Language (UML) to model MAS. Several modeling constructs are introduced including *Agent, Belief, Goal, Plan, FIPA Performative, KQML-Performative*, and *Blackboard*. In addition, the following diagrammatic constructs are introduced: Agent Goal Diagram to model the relationships between the goals and the environment of an agent; Use Case Goal Diagram to model the relationships between use cases and goals; Agent Domain Model to facilitate understanding of domain knowledge of an agent; Agent Sequence Diagram to model interactions within an agent. Similarly, Agent Activity Diagram and Agent Statechart Diagram are introduced. The framework is illustrated by an agent-based intelligent elevator system.

The framework is based on extensions to UML to support multi-agent systems (MAS) development. Their approach is rooted in the BDI formalism [15], but stresses practical software design methods instead of reasoning theories. In particular, we propose to extend UML with modeling constructs called *Agent, Belief, Goal, Plan, FIPA Performative, KQML Performative*, and *Blackboard*. *Agent* is the super-type for all agent types. Belief, Goal and Plan model the reactive and proactive behaviors of agents. An agent has, among other data types, a collection of beliefs, goals and plans. *Belief*s are the agent's observations and/or sensing of the environment and are updated by sensors or other agents. Changes in an agent's beliefs trigger the re-evaluation of the utility values of goals of the agent. Changes to goals' utility values result in pre-empting some plans and initiating new plans. Execution of plans affects the environment, which in turn changes the beliefs, and so on. Agents communicate with each other through agent communication performatives such as FIPA or KQML, or shared blackboards as in Linda or its extensions. In the conceptual model of our framework the Agent Goal Diagram (AGD) is introduced to model the relationships between the goals and the environment, the Use Case Goal Diagram (UCGD) to relate use cases and goals, Agent Domain Model (ADM) to facilitate understanding of agent domain knowledge, Agent Sequence Diagram (ASD) to model interactions within an agent. Similarly, Agent Activity Diagram and Agent Statechart Diagram are introduced.

1.7 Integration of Mobile Agents/Genetic Algorithms

Papavassiliou et. al. [16], present an agent based approach for building a framework where resource allocation is provided under the control of different and often-competing stakeholders (users, network providers, service providers, etc.). They describe the efficient integration and adoption of mobile agents and genetic algorithms in the implementation of an effective strategy for the development of effective market based routes for brokering

purposes (i.e., in the future multi-operator network marketplace). The agent based network management approach represents an underlying framework and structure for the multi-operator network model, and can be used to collect all the required management data. The proposed genetic algorithm provides a kind of stochastic search for optimal resource allocation strategies. [16]

Agent programming was developed in the distributed programming field as a flexible and complementary way of managing resources of a distributed system. Distributing intelligence across the network allows the fast exploitation of advanced services that dynamically adapt to the user's requirements. User requirements are automatically translated into network requirements, and this implicitly assumes the possibility to interact with network equipment. For example, network providers who need application level information to better manage their resources can better satisfy their user needs while minimizing their costs. Moreover, content providers can gain the knowledge of the network resources needed by their services to be properly accessed.

2 Case studies

Lets consider the merits of ABS with regard to their inherent characteristics and properties as they have been studied or applied to three specific problem areas: 1) distributed command and control, 2) information fusion, and 3) data management.

2.1 Distributed command and control

ABSs are particularly suitable for satisfying both functional and nonfunctional DCC requirements, especially in satisfying application scalability, mobility, and security (SMS) expectations. A general set of DCC SW requirements (SRs) was developed based on needs aligned with current computer science technology and inherent limitations [12]. ABS advantages (i.e., SMS) are enabled mainly through a stronger messaging/coordination (MC) model; however, the impact of key DCC system/functional requirements poses the greatest SW challenge. While information fusion, information summary and analysis, and decision support are only tangential to SW technology advances (see Figure 2). Our analysis indicates six key challenges best-addressed using agent technology to provide:

1. Higher-level interfaces to distributed objects,
2. Asynchronous object interaction,
3. Message support for sporadic network connections,
4. Secure object communication and information system operation,
5. Support for richer peer-to-peer programming models,
6. Accelerated SW development productivity.

SW Requirements / **SW Technology Limitations**	Distributed cmptng	Fault Tolerance	Mobile Code	Security	Information Fusion	Information Analysis Summary	Decision Support	SW Productivity
Higher-level Interfaces	X			X				
Asynchronous Interaction	X							
Sporadic Network Support	X	X	X					
Security			X	X				
Peer-to-peer Models	X	X						
SW Productivity								X

Figure 2. A mapping of the SW requirements to the limitations of the current SW technology

ABS is an evolving paradigm that strives to create SW that can mimic certain human behavior. Agents are typically endowed with human-like characteristics. For example, agents are normally considered to be autonomous, adaptable, social, knowledgeable, mobile, and reactive [17]. Lets consider therefore, the comparative benefits of agent technology.

A representative agent architecture by Sycara et al. [18] describes planning, communication and coordination, scheduling, and execution monitoring of agent activities. Agents' access shared information, implemented through a coordination model that can be both domain specific or independent. Griss et al. [19] describes a generalized agent architecture with facilities for locating and communicating with mobile, disconnected agents, and for gathering information about groups of agents. Griss's architecture provides services and support for mobility, security, management, persistence, and the naming of agents.

In general, most agent architectures include support for DCC aspects through a general MC paradigm (i.e., any agent can communicate with one or more agents). This approach encapsulates messages that agents send and receive [17]. OO methods utilize the concept of data encapsulation, which provide for simple SW functions to access an object's data. These functions, not direct data access, are responsible for data retrieval and update. This capability limits the SW (i.e., coupling) that must change due to nonconforming data formats, etc. The agent paradigm extends encapsulation from data to messages sent among agents through an agent coordination model [20]. The model defines how agents communicate among themselves, and can be seen as coordinating communication based on the time a message is sent (temporal) or the names of the target agents (spatial). These models provide the ability for communication that is encapsulated and asynchronous with the use of blackboards, and tuple space models and associated pattern matching, such as Linda [21]. Agents that use a blackboard or Linda type coordination provide a level of indirection for agent communication (i.e., agents post messages to a

blackboard, while subscribers to the blackboard retrieve the message). The agent that sent the message may have no idea who actually receives it. This concept allows for asynchronous and encapsulated communication among a collection of connected or disconnected agents, a capability not currently available in non-agent systems.

Messages are written in an agent control language [22] (ACL) such as KQML or the FIPA ACL, which provide a structured means of exchanging information and knowledge among agents. ACLs support a higher-level communication protocol that does not currently exist for distributed objects. On this basis, lets consider how the DCC concept challenges ABS SW development.

2.1.1 Higher level interfaces to distributed objects

Agent technology, based on a flexible MC scheme and control language, (conceptually) require agents to be connected to blackboards, not other agents[17]. The encapsulation of messages allows for agent interfaces to change, requiring only minor modifications to a blackboard, not to all calling agents. This capability provides for a more robust interface than is currently available in distributed object systems. Moreover, ACLs provide the ability to pass propositions, rules, actions, and states among agents. In this way, messaging is not merely a way of activating a function on a remote host, but provides a way of sending information to another agent. This information can be used to describe what requirements need to be met for an agent to take action, what states the sender and receiver will be in after the action takes place, or what states the agents will be in when the overall transaction is complete [22]. Information sent from one agent to another may also be informative or declarative thereby causing no agent action.

The challenge of implementing such an agent interface is selecting both an MC architecture and an ACL. Currently, no universally accepted MC architecture or ACL means that for an ABS to take advantage of this high-level interface, there must be very specific and precise specifications on how agents will communicate (i.e., using precise ACL syntax).

2.1.2 Asynchronous object interaction

Griss et al. [19] points out that ABS typically have simple interfaces, and derive capability from loose coupling and asynchronous messaging (i.e., messages are sent and retrieved through a loosely coupled temporal agent coordination model). Cabri et al. [20] reference two coordination models that provide asynchronicity. The first coordination model is blackboard-based and provides a shared area where agents' send/retrieve messages. Any authorized agent can read messages posted to the blackboard. Other agents determine whether to retrieve the message based on the sending agent's identifier and therefore knowledge of the agent identifiers is required. The second is based on the Linda coordination model, which defines a messaging protocol, made up of a tuple of information (e.g., a tuple may include the data format, the date of creation, the classification, or a list of keywords). These tuples are placed in a shared area, such as a blackboard. Agents access these messages, not based on agent identifiers, but on a query of the tuple information, (i.e., an agent may retrieve all messages created yesterday with the "Taliban" keyword). This model is asynchronous, and does not require knowledge of the agent identifier.

Both model types are mature and widely used. They provide needed asynchronous behavior but suffer from single-point failure outages. Thus, a single blackboard ABS is exposed to security and performance failures and requires multiple blackboards to provide fault tolerance.

2.1.3 Message support for sporadic networks

One main advantage that ABS provide is flexibility (i.e., ability for agents to change location) along with communication path redundancy. Vogler et al. [23] propose a distributed transaction model using a two-phase commit protocol to verify message delivery. The model must support storage of undelivered messages within the agent, or support the ability to rollback the transaction, if synchronous transactions are required. If a transaction has not completed, then various network/graph theory algorithms can be used to determine a viable path prior to reattempting the transaction. Alternatively, agents can move to another location and try again. If a physical path cannot be found then the transaction is not possible.

Both messaging and mobility can be effectively used to communicate over a sporadic network; however, if the network degrades too much, communication becomes infeasible. Distributed transaction protocols are very useful for verifying the success of transactions, and can be used to ensure network security with the caveat that this capability will limit overall system response time.

2.1.4 Secure communication operations

As Abadi [24] notes, it is practically impossible to construct a truly secure information system. Communications are secure if transmitted messages can be neither affected nor understood by an adversary; likewise, information operations are secure if information cannot be damaged, destroyed, or acquired by an adversary.

Security in a distributed system can be enforced through system wide policies, which are often static, and difficult to modify and enforce [25] ABS can enforce a security policy defining what must be done and what must not be done when information is moved, stored, created, or destroyed. ABS provide multiple, standalone, persistent processes that can act at high speeds to ensure that all rules are always followed. Encapsulated instructions concerning what actions to take under what circumstances enables agents to execute very complex operations, enabling participation in complex collaborative security protocols (e.g., key updating/multiparty authorization).

2.1.5 Peer-to-peer programming models

Fortunately, through the use of blackboard and Linda type coordination models, the programming model of agents can be very general. Any number of agents can send messages

to one or many blackboard(s), and any number of agents can receive messages from one or many blackboard(s). Virtually any topology can be created which allows for very broad scalability of the network. Care must be taken in defining the bandwidth, messaging rates, and processing requirements and will require tuning to enhance fault tolerance and performance.

2.1.6 Increasing SW development productivity

There are indications that agent technology may provide some SW development productivity improvement [19]. While there exists no empirical evidence to support this, the theory claims that ABS increase the level of SW reuse. Agents are SW components that have their messaging, functionality, and location encapsulated thus increasing productivity. Likewise, if standard MC protocols and ACLs can be defined, the agent development teams may require less communication overhead because the interfaces are far richer than with traditional programming.

2.2 Information fusion

In theory, an information SA scours multiple heterogeneous information sources to proactively acquire, semantically understand, process and distribute information and perform other information processing related tasks at the behest and bidding of a specified user. This technology focuses on obtaining a battery of semantic insights from the information-glut/overload that we now face and delivering this semantically digested information in an easy to use/navigate interface.

One so-called Digital Assistant (DA) ABS offers a variety of information gathering/management and processing features where you can: (1) set up a personal watch-list for companies, news and keywords; (2) monitor various online newsgroups and topics of interest; (3) monitor what companies and topics are favored by media; (4) track regular financial data to get a statistical sense of bullish/bearish Sentiment in the market. Results are made available in a decision-ready format (tabular and statistically aggregated percentages) with the flexibility of setting up an email alert containing the digest [26, 27].

The most advanced feature of the DA is an attempt to gauge investor sentiment from various online message boards in the form of an *Opinion Rating*. Various public message boards are scoured to understand what investors are saying about the companies and based on a semantic understanding of these messages a quantitative *Public Opinion* index is formed (assessing the opinion-pulse in the stock markets). Future enhancements could include a news opinion engine that will (at the aggregate level) *understand* what people are saying about a company or how the media is profiling a particular company as well as the ability to query the DA through email. Such enhancements could provide insights into when and by how much market psychology, herd mentality and media exposure has an impact on a stock's price.

The VIPAR project/tool employs ABS technology 1) to utilize the ability for broadcast and peer-to-peer communication among agents, 2) to follow rules outlined in an ontology, and 3) provide persistence (because of the ability for agents to suspend processing on one machine, move to another, and resume processing). These strengths are combined for the purpose of providing an Internet-based DA to support aspects of intelligence, surveillance, and reconnaissance (ISR) in multiple languages[28].

2.2.1 Background

Detailed analysis of large collections of heterogeneous unstructured information is an obvious ISR need[2]. The problem can be viewed in two parts, first how to gather and structure information, and second how to organize and classify information.

2.2.2 Approach

Two broad approaches exist to efficiently gathering and structuring frequently changing heterogeneous Internet accessible information. First, we could obviously use Internet search engines (ISE), which (typically) use programs that recursively traverse links, capturing non-trivial terms on each page. Pages are organized based on the *relevance* of encountered terms enabling a wide variety and number of documents to be categorized according to relevance and made available for further refined searches/reorganization.

ISE weaknesses include 1) existing pages in the system are infrequently re-traversed tending to make the information stale, 2) the Internet pages have no consistent format, and therefore, the semantic content of a page

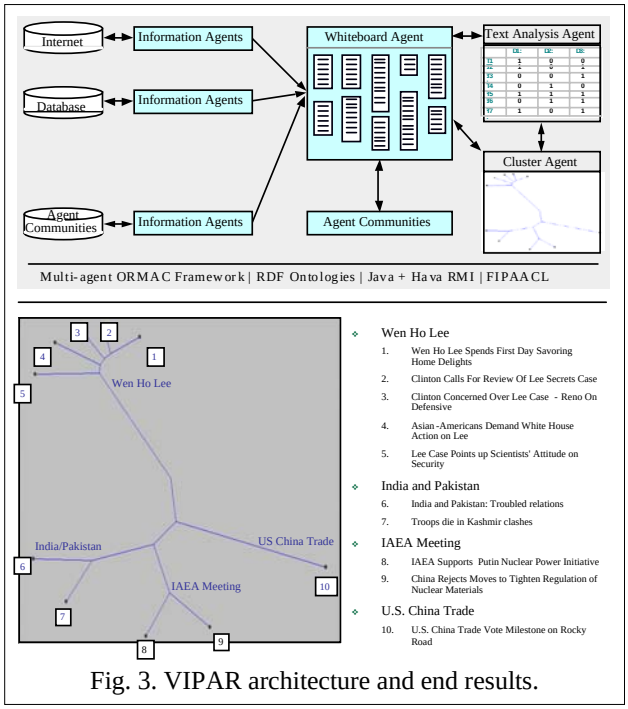


Fig. 3. VIPAR architecture and end results.

[2] Virtual Information Center (VIC) at US Pacific Command, gathers, analyzes, and summarizes information from Internet-based newspapers on a daily basis (a manual, time and resource intensive process).

cannot be easily discerned, 3) the documents are organized based solely on the presence of a keyword in a document (regardless of other attributes like timeliness).

Alternatively, the second approach gathers and structures Internet information using agents. The agents provide various ways to retrieve and organize information, including agents that are capable to access multiple sources, and to filter based on the relevance to the user [18, 29]. Non-cooperating agents perform the information retrieval task, cooperating agents organize the information based on relevance, and finally, adaptive agents deal with uncertain, incomplete, or vague information [30]. Additionally, transforming the inherent and chaotic structure of newspaper articles into a common schema is a difficult problem that must be overcome.

2.2.2.1 VIPAR: unique approach

The VIPAR server uses a set of information retrieval agents to gather news related, non-redundant heterogeneous information from the Internet newspapers, and to format the information using XML (Fig. 3). A whiteboard agent acts as an information-clearing house. Agents submit their articles to the whiteboard agent, who preempts/deletes duplicate articles, archives stale articles (beyond a prescribed age), and feeds articles to agents that have "subscribed" to the whiteboard. A team of cluster agents organizes articles into a vector space model (VSM), then into clusters of articles.

2.2.2.2 VIPAR: information agents

These agents gather and organize information through the transformation of HTML formatted information into XML formatted information. The conversion from HTML to XML is a two-step process. An ontology is defined to provide a common semantic representation and structuring of the heterogeneous information. This ontology embodies the transformation of HTML formatted information to XML formatted information. This ontology is expressed in an XML variant called the Resource Description Framework (RDF, see http://www.w3.org/RDF/). The RDF syntax allows directed graphs to be expressed in an XML-like format. An Internet site is a collection of linked Internet pages. A site is viewed as a directed graph and RDF provides a way to model the linked pages. Furthermore, our agents understand these RDF instructions. A series of RDF ontologies have been developed for the newspapers accessed by the VIPAR system. Each site ontology describes a newspaper: (1) meta-information about the newspaper, and (2) describes site-specific agent actions (e.g., login, etc.). Based on the ontological description of a newspaper site, the agent monitors and manages the information at the site.

An HTML→XML conversion is completed using the defined ontology. An agent, using the RDF ontology, to understand the site layout/semantics can autonomously retrieve articles of interest, and perform the conversion into a structured XML formatted document. Each converted article contains a rich set of XML tags ranging from the time and date the article was discovered, URL location, to the XML tags that format the article. Each agent monitors the site looking for new articles. Fresh articles are formatted and posted to the whiteboard agent.

The ontological site description (OSD) includes a root URL where the agent begins traversal of the site and from which the agent resolves relative site URLs. The OSD includes a series of regular expressions used to describe the table-of-contents for the site. The site description includes a series of regular expressions that describe article pages of interest along with contextual information (i.e., differentiating the text of an article from the myriad of unimportant information (boilerplate, banners, ads, etc). Meta-information is maintained which includes the newspaper's name and the name of the collection under which VIPAR classifies the newspaper, as well as site-specific actions taken by the agents (e.g., search depth limit [hops from the root URL], minutes to wait between rescanning for new articles, etc.).

Using the RDF ontology agents' monitor/manage each site. They check each link against its ontological criteria to discriminate table-of-contents versus article pages. If an article page of interest is found, the agent requests the whiteboard agent verify that the article is not already posted. If the article is not posted, the agent reads the page, distills out clean article text (i.e., filters the raw text from nonessential/extraneous information). The agent marks up the clean text using XML, tagging the parts of the article (title, author, date, location, paragraphs, etc) depending on the site, and then posts the information to the VIPAR whiteboard agent. The agent continues to monitor the site, posting new information of interest as it becomes available. The VIPAR client is also an ORMAC agent that contains a graphical user interface. The client agent communicates with both the whiteboard and cluster agents to direct/refine searches and clustering.

The whiteboard agent maintains all current articles, ensuring no duplicates, and removing articles beyond a certain age. The cluster agent subscribes to the whiteboard agent and thus is notified when an article is added or removed from the whiteboard. When the cluster agent is notified of a new article (as discussed below), it examines the contents of the article and adjusts its search and clustering tables appropriately. Likewise, the tables are adjusted when the whiteboard removes an article.

2.2.2.3 VIPAR: dynamic article clustering

Two basic steps are taken to organize articles into clusters. The first creates a VSM from the articles. The VSM presumes that newspaper articles and their significant terms (words) can be represented as elements of a multi-dimensional vector space. Within this space, each significant term is represented by a new dimension, and a document is represented as a vector within this multidimensional space [31]. The value of each vector coordinate is an entropy-based function of "local" and "global" frequencies of the word corresponding to this

dimension. The cluster agent maintains information containing the frequency of occurrence of terms within a document, called local term frequency, and over the entire set of documents, called global term frequency. These term frequency counts are then used to calculate a weight for each term in each document, which is called the document term weighting.

The second step creates a similarity matrix (SM) that provides a pair wise comparison of each document in the system. We use the dot product (i.e., cosine of the angle between the vector pair) as the measure of similarity between two document vectors. This generates a global SM of size *n* x *n*, where n is the number of documents contained in the document collection. Only the upper triangular portion of this matrix is needed because it is a symmetric matrix. Note, when a document is added or removed the VSM *must* be updated. This is due to the changes in the global frequency of words that are contained in this document. The brute-force approach is to re-compute all the document vectors in the document collection (i.e., document term weights of each document vector) as well as a global similarity matrix. However, the time/space complexity is $O(n \cdot d) + O(n^2)$, where d is the document vector space dimensionality. This is very expensive when the collection size grows. An approach is needed to more efficiently update the SMs. A sliding-window-based approach is used.

The whole SM is modeled as a circular array of size $\frac{n(n-1)}{2}$ with a pointer initially pointing to the first array element. When a new document is added or removed from the collection the p percentage of the SM is updated and the pointer is forwarded $p \cdot \frac{n(n-1)}{2}$ steps from its current position, thus pointing to the next stale entry of the array.

A series of experiments was conducted to determine how changes in global term frequency affect the similarity values. Updating 5% of the global SM every time a single document is added or deleted preserves high accuracy. To compare SMs, several measures were made based on the values of their determinants, traces, and x^2 distribution. In other words, it takes 20 document additions or removals to fully update the SM. This method resulted in acceptable dynamic similarity update performance.

Finally, a global SM is used to perform on-demand clustering of the documents of interest (e.g., the documents retrieved in response to a user query). For a set of documents to be clustered, the local SM is constructed by including the cells of the global SM that in turn corresponds to the documents of interest. This local SM is used to analyze the documents of interest based on their closeness in the document vector space. The documents are merged into clusters using an agglomerative hierarchical clustering algorithm [32]. When all of the documents are combined, a Phylips Tree is generated to illustrate the hierarchical tree structure of the clustered documents (see Fig. 3 lower half). The Phylips Tree (or cluster diagram) is a type of dendrogram. The nodes of the tree represent each article while the edges (or links) between nodes represent relationships. In general, the closer (based on distance and hops) two nodes are, the more similar the articles. If links from two nodes share a vertex, then these articles are the closest in the set of articles. The longer links between article nodes indicate greater dissimilarity.

2.2.3 Results

Organization of the acquired information using the VIPAR system was very successful. In an experiment comparing the organization of news articles done manually, versus organized by VIPAR, results favor VIPAR as the preferred method. The experiment involved searching a collection of newspapers for key terms that produced a number of relevant news articles. This collection of articles was then manually organized based on the contents of the articles. Following this manual process, VIPAR was used to organize the same article set and the results from both methods compared. A search was performed on September 21, 2001, using the phrase “nuclear weapons.” At the time, five newspapers were in the VIPAR system, (1) Japan Times, (2) Pacific Islands Report, (3) Inside China Today, (4) Russia Today, and (5) Sydney Morning Herald. The results produced 10 articles, with various titles (see Fig 3 lower half).

These results are typical of an average search engine, except that VIPAR targets newspapers only and is timelier because it filters out articles older than a few days. Manually clustering these articles put the same articles into the same category. This articles collection covers four broad areas, 1) the Los Alamos Nuclear Scientist Wen Ho Lee, 2) the India and Pakistan conflict spurring nuclear weapons development, 3) an International Atomic Energy Agency meeting dealing with nuclear material, and 4) U.S. China Trade Policy dealing with nuclear material. To manually organize a small number of articles like these can be done fairly quickly by a knowledgeable person. However, as the number of articles *increases* so does the time required to *manually* organize the articles.

VIPAR clustered articles within a few seconds and produced 4 distinct groupings. Fig. 3 (lower half) shows a comparison of the VIPAR cluster to the manual clustering. The four groups determined by VIPAR match extremely well to the four groups of articles manually organized. VIPAR clusters provide an intuitive (i.e., natural, quick and effective) way to organize and visualize this information.

2.3 Data management

This case study uses SAs to divide and concur massive amounts of distributed data. The SAs, which run on the machines where the data resides, collaborate to produce movies from the requested data, which are sent back to the remote client for display. The quality of the movies can be varied depending on the available network bandwidth.

2.3.1 Background

Simulationists who model physical phenomena commonly deal with massive (terabytes) datasets widely distributed and derived from months of supercomputing. Refining these models and algorithms to maturity requires numerous iterations where the scientist modifies the algorithm, and validates the resulting output. The scientist either examines the candidate dataset in raw form or invests considerable time and effort to analyze the data using highly specialized hardware and software tools.

2.3.2 Approach

We proposed a large system of distributed SAs spread over the distributed data as a simple and flexible way to help scientists validate simulation results and refine the simulation model/algorithm. We used 100 time steps of data from a supernova simulation segmented into 800 individual pieces, managed by 800 agents, running on conventional systems. We have developed a system where a single software agent is responsible for each individual segment of data. Upon request, these 800 agents work together to produce a visual representation as shown in Fig. 4. Our results illustrate that a large system of software agents is a simple and flexible solution to the problem of data validation during the development of scientific simulation models. In work with numerous scientists at various laboratories and universities, we have been successfully using this approach to render data from a supernova simulation.

The agent architecture of this experimental system involves multiple software agents, each of which has one of three basic tasks. The first type of agent is called the data controller agent. The data controller agent monitors the simulation output directory for newly created data files. When one is found, this agent then creates and assigns eight new data agents to eight equal sub-cubes of the new file. These data agents are the second type of agent used in the system. The creation of eight agents per new file is arbitrary and easily changed. Each of these new agents is then responsible for fielding requests from other agents. The typical request is for an agent to provide an image from an XY plane of data under its control. In this case, the individual agents will generate an image of a 2D plane from the 3D sub-cube that they are responsible for. If the requested plane falls outside of this cube, the agent ignores the request. The agents also have the ability to vary the quality of the images produced. A blackboard is used to collect images from various responding agents. From this blackboard, the third type of agent, the movie producer agent, assembles the images into a movie that shows an XY plane through the 100 time steps of data. Using different video compressors and decompressors (CODECs) allows the movies to be produced at different detail levels (See Fig. 4 top half).

2.3.3 Results

The dataset was provided by the DOE's Terascale Supernova Initiative (TSI) project. We used a portion of the TSI supernova simulation data to demonstrate that a system made up of a large number of SAs is a viable solution. The original data contains 192 times steps. Each time step containing data from 5 variables, X, Y, and Z velocities, pressure, and density. Data from each variable is represented in a 320 x 320 x 320 matrix of floating point values stored in Hierarchical Data Format (HDF) 4 format in 960 files requiring approximately 128GB of storage. For demonstration purposes, we chose 100 time steps of Y velocity data showing significant activity. This equates to 100 files, each 133 MB stored on two separate PCs.

3 Discussion and conclusions

Lets briefly review our conclusions from the three case studies described here.

3.1 Distributed command and control

A comparison of DCC functional requirements with the capabilities of existing SW technology reveals the limitations of low-level interfaces, synchronous interactions, and requirements for continuous network availability, limited redundancy, and limited productivity improvements. Current technology would require major enhancements (if even feasible) to enable the DCC concept. Moreover, the main strength provided by ABS is derived from the MC model thereby supporting a more flexible and consequently more robust programming model. The intellectual integrity and congruency gained by

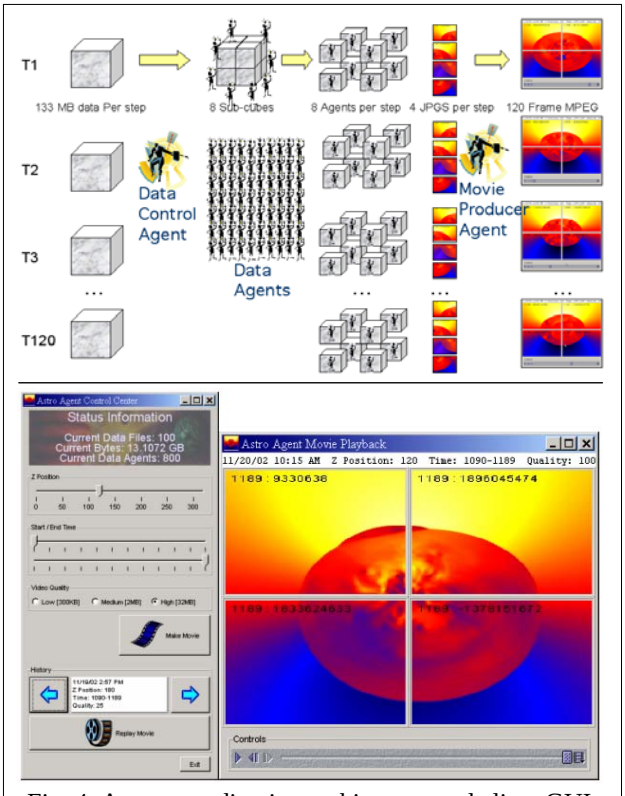


Fig. 4. Agent coordination architecture and client GUI.

mapping the DCC requirements onto the ABS model gives a compelling and natural consistency. Furthermore, ABS can support the DCC functional requirements including security, information analysis/summary, and decision support, but the technology does not explicitly provide these capabilities, and these are challenging problems.

3.2 Information fusion

The ISR/VIC problem involves gathering/analyzing more information that can be reasonably accomplished manually (i.e., the common information-glut/overload dilemma that promises to worsen). To address this challenge, the multi-agent VIPAR system was developed using software agents to retrieve, organize, and graphically present Internet-based newspaper information comparable to that accomplished by human intelligence analysts. VIPAR extends the field of agent technology through the use of a flexible RDF ontology for managing information including the capability to dynamically add and cluster new information entering the system.

Agent technology is well suited to this type of problem for three main reasons. First, the communication mechanism allows for broadcast *and* peer-to-peer message passing. Second, using an external ontology allows for an easily maintainable and/or replaceable mechanism for adapting to an open/changing information environment and rules (intelligence needs). Consequently, agents can be redirected without the need to modify code. Finally, agents are mobile, a natural solution to the needs of intelligence gathering. The ability for agents to suspend operations, move to another computer, and resume operations on command provides for various design/implementation options needed for rapid deployment.

3.3 Data management

A multi-agent system (MAS) for analyzing massive scientific data was developed successfully as flexible and economical solution for distributed data management. Agents monitor the output of a simulation model/run. Anytime the simulation produces new data, the primary monitoring agent logically divides the new data into pieces and creates a new agent for each piece of the new data. Each agent responds to queries about the piece of data that they are responsible for.

In collaboration with scientists from various labs and universities, this approach has been used to render data from a supernova simulation experiment under development. Using 100 time steps of data segmented into 800 individual pieces, managed by 800 agents, running on conventional systems, the agents work together to produce a visual representation of the dataset (Fig. 4). The results indicate that a large system of software agents spread over the candidate dataset can be an adaptable and cost effective method to aid scientists with validating the dataset.

4 References

1. Patel, M., Forward: Advances in the Evolutionary Synthesis of Intelligent Agents, 1st ed. Advances in the Evolutionary Synthesis of Intelligent Agents, ed. P.J. Mukesh, V. Honavar, and K. Balakrishnan. 2001, Cambridge: MIT Press. 480 pages.
2. Subrahmanian, V.S., et al., Heterogenous Agent Systems, 1st ed. 2000, Cambridge: MIT Press pages.
3. Demarco, T. and P.J. Plauger, Structured Analysis and System Specification. 1985, New York: Prentice Hall. 352 pages.
4. Sheldon, F.T., K. Jerath, and H. Chung, "Metrics for Maintainability of Class Inheritance Hierarchies," Jr. of Software Maintenance and Evolution, 2002. 14(3): pp. 147-160.
5. Booch, G., Object-Oriented Design with Applications, 2 ed. 1991, Redwood City: Benjamin/Cummings. 608 pages.
6. Chin, R.S. and S.T. Chanson, "Distributed, Object-Based Programming Systems," ACM Computing Surveys, 1991. 23(1): pp. 91-124.
7. Thorn, T., "Programming Languages for Mobile Code," ACM CS 29, No. 3 (1997)." ACM Computing Surveys, 1997. 29(3): pp. 213-239.
8. Kim, H.Y., Jerath, K. and Sheldon, F.T., Assessment of High Integrity Components for Completeness, Consistency, Fault-Tolerance and Reliability, in Component-Based Software Quality: Methods and Techniques, A. Cechich, M. Piattini, and A. Vallecillo, Editors. 2003, Springer-Verlag: Heidelburg. pp. 259-86.
9. Geihs, K., "Middleware Challenges Ahead," Computer, 2001. 34(6): pp. 24-31.
10. Potok, T.E., N.D. Ivezic, and N.F. Samatova. "Agent-based Architecture for Flexible Lean Cell Design, Analysis and Evaluation," in Working Conf. on Design of Info. Infrastructure Sys., Melbourne Australia: Kluwer, 2000, pp. 181-8.
11. Ivezic, N., T.E. Potok, and L. Pouchard, "Multiagent Framework for Lean Manufacturing," IEEE Internet Computing, 1999. 3(5): pp. 58-9.
12. Potok, T.E., Phillips, L., Pollock, R., Loebl, A. and Sheldon, F.T. "Suitability of Agent-Based Systems for Command and Control in Fault-tolerant, Safety-critical Responsive Decision Networks," in ISCA 16th Int'l Conf. on Parallel and Distributed Computer Systems (PDCS), Reno NV: ISCA, 2003.
13. Kavi, K.M., M. Aborizka, and D. Kung. "A framework for the design of intelligent agent based real-time systems," in Proc. 5th Int'l Conf. on Algorithms and Architectures for Parallel Processing, Beijing: IEEE CS, 2002, pp. 196-200.
14. Kavi, K.M. and H.B. D.C. Kung, G. Pandcholi, M. Kanikarla and R. Shah. "Extending UML to modeling and design of multi agent systems," in

Proc. of 2nd Intl Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS collocated with ICSE03), Portland: Springer, 2003.
15. Rao, A. and M. George. "BDI agents: From theory to practice," in Proc. First Int'l Conf. on Multi-Agent Systems (ICMAS-95), San Francisco: AAAI Press, 1995, pp. 312-319.
16. Papavassiliou, S., et al. "Integration of Mobile Agents and Genetic Algorithms for Efficient Dynamic Network Resource Allocation," in Sixth IEEE Symp. on Computers and Communications (ISCC'01), Hammamet, Tunisia, 2001, pp. 456-63.
17. Jennings, N.R., K. Sycara, and M. Wooldridge, "A Roadmap of Agent Research and Development," Jr. of Autonomous Agents and Multi-Agent Systems, 1998. 1(1): pp. 7-38.
18. Sycara, K., et al., "Distributed Intelligent Agents," IEEE Expert, 1996. 11(6): pp. 36-46.
19. Griss, M.L. and G. Pour, "Accelerating Development with Agent Components," Computer, 2001. 34(5): pp. 37-43.
20. Cabri, G., L. Leonardi, and F. Zambonelli, "Mobile-agent Coordination Models for Internet Applications," Computer, 2000. 33(2): pp. 82-9.
21. Gelernter, D. and N. Carriero, "Coordination Languages and Their Significance," 1992. 35(2): pp. 96-107.
22. Labrou, Y., T. Finin, and Y. Peng, "Agent Communication Languages: The Current Landscape," IEEE Intelligent Systems, 1999. 14(2): pp. 45-52.
23. Vogler, H., T. Kunkelmann, and M. Moschgath. "An Approach for Mobile Agent Security and Fault Tolerance using Distributed Transactions," in Int'l Conf. on Parallel and Distributed Systems, Seoul: IEEE, 1997, pp. 268-74.
24. Abadi, M., "Secrecy by Typing in Security Protocols," Jr. of the ACM, 1999. 46(5): pp. 749-786.
25. Liu, Z., et al. "Pluggable Active Security for Active Networks," in IASTED Proc. Int'l Conf. PDCS, Nov. 2000," in Int'l Conf. PDCS, Las Vegas: IASTED, 2000.
26. K-Praxis, "SonicBoomerang: Semantic Prime-Time for Intelligent Information Agent Technologies." 2003, K-Praxis, http://www.k-praxis.com/archives/000036.html.
27. CCNMatthews, "Intelligence Gathering Service Wins Information Highways Magazine's 2002 E-Content Innovation Award." 2003, CCNMatthews: Toronto, http://www.ccnmatthews.com/scripts/headlines1.pl.
28. Potok, T., Elmore, M., Reed, J. and Sheldon, F.T. "VIPAR: Advanced Information Agents Discovering Knowledge in an Open and Changing Environment," in SCI 2003 Proc. 7th World Multiconference on Systemics, Cybernetics and Informatics (Special Session on Agent-Based Computing), Orlando: IIIS, 2003.
29. Mladenic, D., "Text-learning and Related Intelligent Agents: A Survey," IEEE Intelligent Systems, 1999. 14(4): pp. 44-54.
30. Klusch, M., "Information Agent Technology for the Internet: A Survey," Data & Knowledge Engineering, 2001. 36.
31. Samatova, N.F., T.E. Potok, and M.R. Leuze, "Vector Space Model for the Generalized Parts Grouping Problem," Robotics and Computer-Integrated Manufacturing, 2001. 17(1-2): pp. 73-80.
32. Anderberg, M.R. "Cluster Analysis for Applications," in Probability and Mathematical Statistics, 19, New York: Academic Press, 1973.

Unifying the Interpretation of Redundant Information

Rocchi Paolo
IBM, via Shangai 53, Roma Italy
paolorocchi@it.ibm.com

Keywords: Redundancy, control, information theory, reliability theory

Received: November 12, 2002

This paper discusses the possibility of interpreting redundant information beyond the particular views emerging in specialist sectors. We introduce a theoretical framework that aims at unifying and calculating the main features of redundant information. This theoretical layout has been introduced in professional tuition.

1 Introduction

People tried to handle redundant information from immemorial times. For example, copyists introduced several abbreviations and writing simplification in order to reduce the language redundancy.
Specialists did not tackle redundancy by rigorous methods until the early twentieth century when telegraph and telephone networks, radio emitters began to connect towns, then nations and continents. Infrastructures for telecommunication involved heavy investments and economic pressures drove engineers to optimize the use of these facilities. H. Nyquist and others started to search for optimal transmission and finally C. Shannon established the fundamental laws of data compression and marked the birth of the information theory [1].
These authors accomplished their purposes and aided the progress of technology, but the thorough comprehension of redundancy still remains an open question. Writers brought this problem to light nearly fifty years ago [2] [3] [4], although a formal theory on redundancy does not seem to attract mathematicians' attention so far. The debate remains on the philosophical plane, for example see the initial "Theory of Redundancy" and the next "Deflationary Theory of Truth" [5]. Modern advances in computer science, especially in the Internet, press toward the rigorous comprehension of the different forms of redundancy. We select three essential points from the queries that thinkers have raised.

a) Redundancy regards any kind of information and we question whether results pertaining to the binary technology may be extended to other forms of information [6]. The evidence should prove the contrary.

b) The entropy and the redundancy factor quantify redundancy of digital information. As they are logically disparate, we should integrate them into a comprehensive notion expressed by the mathematical language.

c) Redundancy increases the reliability of data during transmission and storing and also improves the machinery reliability. The relationship between redundant codes and redundant systems should be fully clarified in order to achieve the general and exhaustive knowledge of redundancy.

I was persuaded that these ample themes should be handled within a unifying logic and have driven a theoretical research for years. This paper sets out some results and tries to answer the above points.

2 Redundancy

Let the set $\{\varepsilon\}$ include the entities $\varepsilon_1, \varepsilon_2, \varepsilon_3, \ldots \varepsilon_n$. We assume that two pieces of information are the entities ε_i and ε_j that have the property of being distinct

$$\varepsilon_i \gamma \varepsilon_j \qquad i, j = 1,2,..n \quad (2.1)$$

The item of information ε_x stands for something and we assume that the meaning of information is the function μ of representing α.

$$\varepsilon_x \xrightarrow{\mu} \alpha \qquad x = 1,2,..n \quad (2.2)$$

We could say that μ is the main job of ε_x or, in other terms, ε_x works as a model.
The statements (2.1) and (2.2) formalize two ideas universally shared in current literature. Notably they establish that information is distinct and has semantic properties. Discrete formalism is usual in information

technology (IT) and the pair (2.1) and (2.2) follows this vein.

The word "redundant" derives from Latin and hints something abundant and repetitive with respect to its use. I put forward the following definition of the redundancy in accordance to this naïf idea.

Definition 2.1: *The set* $\{\varepsilon\}$ *is minimal when the number of informational entities is equal to the number of the objects to be represented*

$$n = n_\alpha \tag{2.3}$$

It is insufficient when

$$n < n_\alpha \tag{2.3 bis}$$

When

$$n > n_\alpha \tag{2.3 tris}$$

The set $\{\varepsilon\}$ *is redundant, notably the information surplus provides the redundancy of* $\{\varepsilon\}$

$$R = n - n_\alpha \tag{2.4}$$

The more R is high and the more $\{\varepsilon\}$ is redundant. Redundancy is null if (2.3) is true and is negative if (2.3bis) is true. In substance R gives the excess (when positive) and the lack (when negative) of the models ε. Redundancy is null when the representations are just enough. For ease, the picture of the car in the web page and three phrases invite the web-visitor to buy the car. Four pieces of information have the same promoting significance and make the message redundant

$$R = 4 - 1 \tag{2.5}$$

Any item of information may be modeled as an algebraic entity and definition (2.4) begins to respond to the question b).

3 Methods

Engineers follow two major approaches in order to ensure the reliability of a system. The former provides remedies after the failure has occurred. The latter method is precautionary and precedes the damage

Method (1): *Repairs the failure.*
Method (2): *Prevents the failure.*

Redundancy is a precautionary solution against information failures and falls into (2) [7]. IT specialists preliminarily take care of errors, noise and random irregularities, which will injure transmission and storage. When the information set $\{\varepsilon\}$ is redundant, (2.3ter) yields two possibilities
Method (2.1)*: All the items of* $\{\varepsilon\}$ *are used as models.*
Method (2.2)*: R items are not used as models.*

We detail these either-or ways.

Method (2.1) - When (2.3ter) is true and all the items of information are used, at least one object α has k models

$$k \geq 2 \tag{3.1}$$

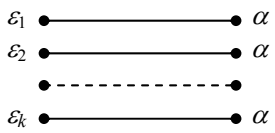
Let P the probability of altering one piece of information, the probability of altering k items of information is

$$P_k = P^k \tag{3.2}$$

As P is lower than unit, k pieces of information, which stand for one entity α, are more reliable than only one item of information

$$P_k < P \tag{3.3}$$

This graph, derived from (2.2), visually evidences how k items of information representing the same object are similar to k units working in parallel



(3.4)

Expressions from (3.1) to (3.3) are formally symmetrical to the formulas that calculate k machines in parallel and they reach the same conclusions [8]. In short, *Method* (2.1) answers point c). The present theory shrinks the gap between the information theory and the reliability theory.

Method (2.2) - When (2.3ter) is true and R pieces of information are unused, $\{\varepsilon\}$ splits into two separate subsets

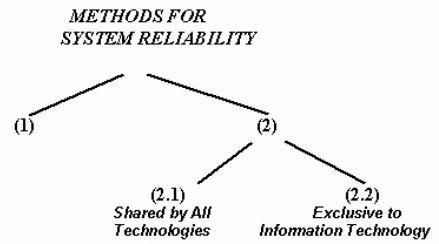
$$\{\varepsilon_u\} \cap \{\varepsilon_z\} = \varnothing \tag{3.5}$$

The subset $\{\varepsilon_u\}$ includes n_α pieces of information with precise significance. The subset $\{\varepsilon_z\}$ has R meaningless items; hence the entities differ from the semantic viewpoint

$$\varepsilon_u \neq \varepsilon_z \tag{3.6}$$

They apply (2.1) thus engineers can follow a special method, which is exclusive to IT and cannot be compared elsewhere. They prepare the subset $\{\varepsilon_u\}$ which is meaningful and $\{\varepsilon_z\}$ meaningless. If the control detects the unmeaning piece ε_z, it reveals an error. This

technique, based on the inequality (2.1), is exclusive to information, while *Method* (2.1) is universal.



This framework clarifies the relations existing between the information sector and other engineering fields, and elucidates point c).

4 Redundant codes

We restrict our attention to *Method* (2.2) and in particular examine the redundancy of codes. We assume the length L is fixed for the sake of simplicity. The combinatorial calculus provides the ensuing result

$$n = B^{L} \tag{4.1}$$

Where B is the base of the code $\{\varepsilon\}$. Now we consider the *minimal code* $\{\varepsilon_\alpha\}$ that, in accordance to (2.3), is the set of codewords just sufficient to represent n_α objects. The base B and the minimal length L_α allow us to calculate n_α. This quantity along with (4.1) makes explicit the redundancy (2.4)

$$R = B^{L} - B^{L\alpha} \tag{4.2}$$

As B exceeds the unit, $\{\varepsilon\}$ is redundant if and only if L is larger than the minimal length L_α

$$R > 0 \quad \omega \quad L > L_\alpha \qquad B f 2 \tag{4.3}$$

This result proves that the redundancy of a digital code relies on its length.

1] Expression (4.3) suggests to calculate the *digital redundancy* R_D of $\{\varepsilon\}$ by the difference of lengths

$$R_D = L - L_\alpha \tag{4.4}$$

2] The *redundancy factor* R_C of the code $\{\varepsilon\}$, already in use, relates the length L with respect to the minimal length

$$R_C = \frac{L}{L_\alpha} \tag{4.5}$$

We make explicit L and L_α with combinatorial calculus and we put them into (4.5)

$$R_C = \frac{L}{L_\alpha} = \frac{\log_B(n)}{\log_B(n_\alpha)} = \log_{n\alpha}(n) \tag{4.6}$$

This result evidences that *Rc* depend on n and n_α. This property also regards R_D that has the same variables of *Rc*. Both of them are coherent with (2.4) in point of mathematics. They differ on the practical plane: R_D and *Rc* regard digital information instead R has general usage. This theory brings to light the relations existing between various measurements of redundancy and clarifies point b).

The base B and n_α objects are usually given in the professional environment, hence the length L_α is the essential reference for digital calculations. Combinatorial calculus provides this result

$$L_\alpha = \log_B(n_\alpha) \tag{4.7}$$

That although neglects the frequency of the codeword. Shannon has the merit of discovering the accurate value of L_α and calculating it by means of the entropy

$$H = -C \sum_{i}^{n\alpha} P_i \cdot \log_B(P_i) \tag{4.8}$$

Where C is a positive constant and

$$\sum_{i}^{n\alpha} P_i = 1 \tag{4.9}$$

H provides the rigorous minimal length and brings evidence of the roughness of (4.7). In fact, if the codewords of the code $\{\varepsilon\}$ are equiprobable

$$P_i = 1 / n_\alpha \tag{4.10}$$

The entropy equals to (4.7) up to the constant C

$$H = -\sum_{i}^{n\alpha} 1/n_\alpha \log_B(1/n_\alpha) = \\ = \log_B(n_\alpha) \tag{4.11}$$

May be proved that this result is the maximum of *H,* when n_α and B are given [1]. In short the Shannon entropy provides L_α in general, while (4.7) is true only if the codewords are equiprobable.

5 Technical refinements

Let codewords be duplicated, tripled etc.

$$R_C \quad f \quad 2 \tag{5.1}$$

High redundancy entails high reliability because the detection of errors is immediate. Per contra volumes are bulky and this range confines the problem

$$R_C < 2 \qquad (5.2)$$

The small number of characters hinders the detection of errors and engineers refine *Method* (2.2) by means of the algorithm, which enhances the control.
This solution although does not rule out the possibility of an input unrecognizable by the algorithm. The generic codeword may be modified during the transmission or the storing to the extent that it could neither belong to $\{\varepsilon_u\}$ nor to $\{\varepsilon_z\}$. In this case, the *Method* (2.2) flops.
Engineers cure the problem and state that the subsets $\{\varepsilon_u\}$ and $\{\varepsilon_z\}$ be mutually exclusive. Using the set theory we write the following constraint

$$\{\varepsilon\} = [\{\varepsilon_z\} 3 \{\varepsilon_{z\mathbf{C}}\}] \, 4 \, [\{\varepsilon_u\} 3 \{\varepsilon_{u\mathbf{C}}\}] \qquad (5.3)$$

Where $\{\varepsilon_{z\mathbf{C}}\}$ and $\{\varepsilon_{u\mathbf{C}}\}$ are the complementary subsets of $\{\varepsilon_z\}$ and $\{\varepsilon_u\}$. As (3.5) is true we have

$$\begin{aligned} \{\varepsilon_u\} 3 \{\varepsilon_{z\mathbf{C}}\} &= \{\varepsilon_u\} \\ \{\varepsilon_z\} 3 \{\varepsilon_{u\mathbf{C}}\} &= \{\varepsilon_z\} \end{aligned} \qquad (5.4)$$

And finally we get

$$\{\varepsilon_u\} 4 \{\varepsilon_z\} = \{\varepsilon\} \qquad (5.5)$$

This equation along with (3.5) establishes the *Excluded Middle Principle*. The bits and the binary codewords comply with this constraint. A binary word is necessary included either in the subset $\{\varepsilon_u\}$ or in $\{\varepsilon_z\}$. Specialists elaborate the most advanced control techniques thanks to this special property [9] which provides the answer to question a).

6 Conclusions

Some authors pursue complex studies about redundancy on the philosophical plane but the conclusions appear generic to engineers. I have searched for the replies to the initial queries by means of the mathematical language and believe that this feature may be appreciated.
These pages stage the examination of redundancy and progressively come from the most ample themes to the specialist ones. In detail:

- This paper proposes the redundancy definition (2.4) which relates the physical nature of information to the semantics. This unitary approach gives an answer to point b).
- This work puts forward the redundancy R_D of digital words which is verbally expressed so far, and relates it to R_C and to R. These discourses try to clarify point b).
- Equation (3.3) and scheme (3.4) elucidate the links between the informational redundancy and the reliability theory as point c) demands.
- The *Excluded Middle Principle* (5.5) and (3.5) explain why technicians can develop very sophisticate binary solutions as point a) presumes.

The cultural meaning of the present work has proved to possess valid educational qualities. They have been partially taught in high schools and in basic training in IBM.
An ample theory on information, systems and control includes the equations presented in this paper [10] and this is the last feature, which I aim at highlighting.

References

[1] Shannon C., Weaver W. (1949) - *The Mathematical Theory of Communication* - Univ. Illinois Press, Urbana.

[2] Calderbank R. ed. (1996) - *Different Aspect of Coding Theory* - American Mathematical Society.

[3] Kriebel H.C. (1965) - *A Resume of Mathematical Research on Information Systems*. - Carnegie Institute of Technology, Pittsburg.

[4] Cherry C. (1996) - *On Human Communication: a Review and a Criticism* - MIT Press, Cambridge.

[5] Field H. (1986) - The Deflationary Conception of Truth - in MacDonald G and Wright C. (eds.) *Fact, Science and Morality,* Blackwell, Oxford.

[6] Marin L. (1994) - *De la Representation* - Gallimard, Paris.

[7] Ramakur R. (1993) - *Reliability Engineering: Fundamentals and Applications* - Prentice Hall, N.Y.

[8] Shen K., Xie M. (1990) - On the Increase of System Reliability by Parallel Redundancy - *IEEE Transactions on Reliability* vol 39, n.5.

[9] Wakerly J. (1978) - *Error Detecting Codes, Self-checking Circuits and Applications* - North-Holland, Amsterdam.

[10] Rocchi P. (2000) - *Technology + Culture = Software* - IOS Press, Amsterdam.

Assessing the Potential Impact of an Electronic Grade System to the School Environment

Eva Jereb, Teja Toman
University in Maribor, Faculty of Organisational Sciences, Kidričeva cesta 55a, 4000 Kranj
eva.jereb@fov.uni-mb.si, http:// www.fov.uni-mb.si/eva

Keywords: internet, modelling, electronic grade system

Received: February 7, 2003

Nowadays the tempo of modern life is very fast which prevents parents from their regular contact with schools. We have developed an application, more precisely, an electronic grade (e-grade) book which enables the parents to monitor and control their child's education. By entering the username and password the parents would have an insight into their child's grades, inexcusable absence, test dates, cultural, sport days, natural science days, teachers' notes to the parents and so on. This paper shows an example of an e-grade book on the internet as well as the advantages and disadvantages a possible introduction of an e-grade in primary schools might bring about. Teachers' and parents' response to a possible introduction of an e-grade book is also presented.

1 Introduction

The age of computer technology is directed toward a simplification of everyday work. With the help of computer-communication technology, which has a significant social and economic impact in the last 20 years (Adams and Warf, 1997), we administer the information on the basis of which we decide and look for solutions of different problems which we encounter every day. We want to avoid the unnecessary paper work, which is possible through different informational systems directed towards automation of bureaucracy. And the education sector has not been immune to the impact of these developments as well (Mitchell and Hope, 2002).

A current question in Slovenia is how to improve the management at schools from the point of view of the informatization as well as from the point of view of the so called 'upgrading' of a an classic grade book with an e-grade book. To get the best possible results a number of projects is being carried out by different performers.

We have decided to elaborate an information system for monitoring primary school pupils' results – an e-grade book on the internet. We have also carried out a survey (among parents and teachers) in four primary schools in Gorenjska, which momentarily do not use such an informational system. The main aim was to find out how familiar they are with it and what their relation towards the new technology is.

2 A Design Of Informational System For Monitoring Primary School Pupils' Results - An E-Grade Book

Informational system for monitoring primary school pupils' results is entirely written for the internet surrounding in programme languages HTML (HyperText Markup Language), JavaScript, VBScript, Active Server Pages (ASP), Active Database Objects (ADO) (see Sussman and Homer, 1998), and takes the data from MS SQL data base, version 2000 (more about see Schofield, 1994; Reisman, 1994; Cooper, 1997; Šalomon, 1998).

The web application E-grade book consists of four parts. The first one is designed for entering look-up tables as well as data compiling which is only accessible for the administrator of the system. The last thing we would want is the data to be updated by the people who are not qualified to do this work. The next part is an e-grade book itself that is divided into two parts: the first one is designed for teachers only, namely for entering the grades, notes to the parents; the second part is designed for the parents' insight into their child's grades. The third part comprises of notes for the parents that is the same for the entire class. The last, fourth part enables the other teachers the insight into the e-grade book.

The first page of the application includes the title of the application and a primary school logo. Below are also two connections to the look-up tables and to the e-grade book itself.

After entering the username and password and clicking the 'Confirm' button the application first checks the database for the existence of the entered username. Then it enciphers the password on the system code 'MD5', compares it to the one in the data base and finally checks if the entered group containing also the user has the access to the required page (Papa et al., 1999). If all the conditions are accomplished the user can continue, otherwise the system prevents him/ her from doing so. The application in use is also going to be protected on the level of internet with the help of a certificate and key

code (e.g. Verisign – coding of the internet page with a help of a public and personal key).

3 The Operation Of An E-Grade Book

3.1 A Part Of An E-Grade Book Designed For Teachers

The first part of an e-grade book is designed for teachers' use only. Anyone having an access to the e-grade book must have his/her username and password that are both appropriately protected against possible breaks into a computer system. Before its use the teacher must register through the registry screen.

If the username or the password has been entered incorrectly three times in a row we are notified about it and the application returns to the basic choice. The second or main part of the application which includes modules for both entering the grades as well as the parents' insight into their child's grades, identifies itself on the basis of the entered username and password whether the user is a teacher or a parent and the application then shows the appropriate screens. After the registration the teacher has to choose the class and the subject for which he/she will enter the grades. He/she can only choose among the classes and subjects he/she teaches which has to be previously defined by the administrator of the system in the look-up table.

After choosing the appropriate elements the teacher has to decide which pupil will receive a grade. The procedure stays the same when the teacher wants to mediate a notice, a comment on an individual pupil to his/her parents. The choice of an individual pupil (as well as a class and a subject on a previous screen) follows the same drop down menu lists, which can be evident from the screen pictures.

The teacher has thus reached the main screen of an e-grade book. In the upper part of the copy he/ she can follow the previous success in a particular subject, but he/she also has a choice to enter a new grade or a note for the parents (see Figure 1). It should be mentioned that it is impossible to correct or delete the previous grades for safety reasons. This can only be done later with the mediation of the administrator (more about see Amundsen, 1999).

An e-grade book has been designed in such a way that a teacher can perform all the transactions already in the classroom provided that classrooms all have the appropriate computer equipment.

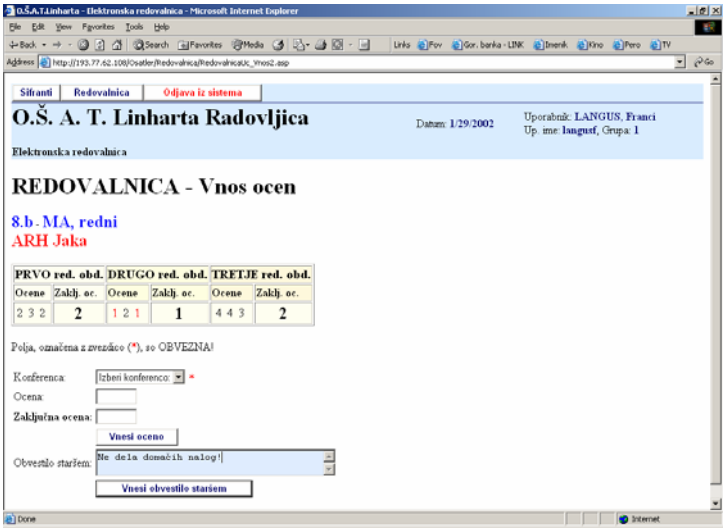


Figure 1: Entering grades or notes for the parents

3.2 A Part Of An E-Grade Book Designed For Parents

A parent with the acquired data on username and password can link to the school web page, register him-/herself to the system and choose a child whose e-grade book he/she would like to see. It needs to be mentioned that in a drop down menu list there is only a child or children of parents who registered in the system.

The next screen shown to the parents is an e-grade book of a chosen child. Here the parents can obtain all the grades with final grades for all the subjects and at the same time follow the teacher's notes (see Figure 2). Every note includes a name of the teacher (in brackets) who wrote it. Notes are written in different colours (red- more important, blue- less important) and for the sake of easier inspection negative grades are also written with a different colour (red).

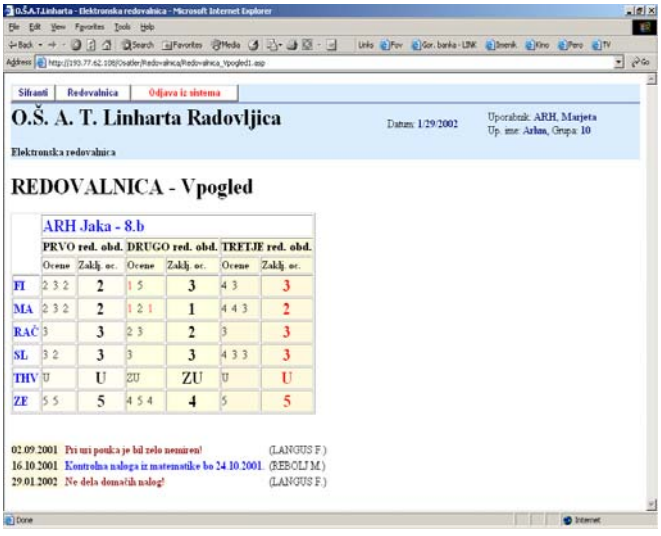


Figure 2: Insight into an e-grade book of an individual pupil

The same applies for the copy of an e-grade book for the entire class that is only intended for the teachers. A teacher first selects the appropriate class and after a click on the button the entire e-grade book is presented.

3.3 A Part Of An E-Grade Book Designed For The Administrator

Let's stop also at the part of an e-grade book designed for the administrator of the system. Before the application is used a look-up table and a database have to be completed with teachers, pupils, classes, parents and so on. This part of the application is of the utmost importance and a special caution is needed for the entire system of an e-grade book not to collapse with the inappropriate data (see Figure 3).

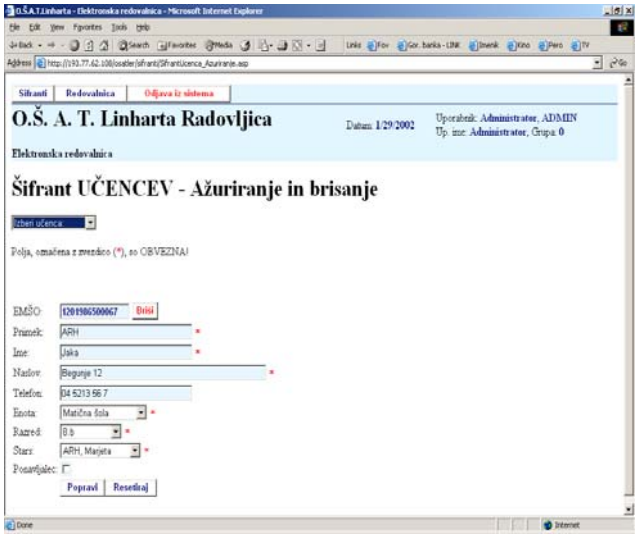


Figure 3: Picture of one of the look-up table – a look-up table of a pupil

4 Responses To A Possible Introduction Of An E-Grade Book In Primary Schools in Slovenia

To get the information on the opinions and views of possible introduction of an e-grade book of both primary school teachers and parents we have carried out a survey and discussed some problems accompanying a possible introduction. A survey has been carried out in A. T. Linhart Primary School Radovljica, Fran Saleški Finžgar Primary School Lesce, Primary School Gorje and Primary School Žirovnica. 92 teachers and 290 parents participated in a survey. The questionnaire for parents (see Appendix A) has been distributed between the pupils from 1st to 8th grade and the classes have been chosen randomly. In the introduction the questionnaire included the very purpose of the survey and a short description of an e-grade book on the internet. The questionnaire for teachers is shown in Appendix B.

4.1 Teachers' Response

Less than half of the teachers were inclined to possible introduction of an e-grade book (43%). The majority was against it with the explanation of the e-grade book being only an additional work for the teacher and a loss of time (33%), another 22% explained that nothing has to be modernised. And some of them did not even know what the question was all about (2%). The teachers' inclination towards the introduction of an e-grade book is shown in Figure 4.

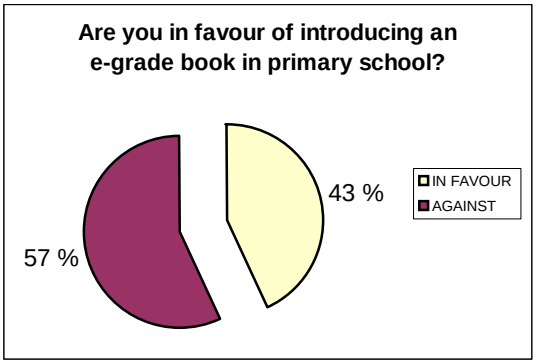


Figure 4: Review of teachers' inclination towards introduction of an e-grade book

With a possible introduction of an e-grade book a relatively small percentage of teachers (21%) was of an opinion that there would be no problems with operating with the programme with a preliminary seminar on how to use the programme. Some teachers (40%) thought that possible introduction of an e-grade book might negatively affect the relation between teachers and parents because a personal contact would no longer be needed. Some also worry about the programme to be too complicated (29%), and the minority (10%) thinks that numerical grades are not enough an information.

The majority of teachers (43%) think that in a case of a possible introduction of an e-grade book parents would come to school on less frequent basis than now (all the information could be obtainable on the internet), some (37%) think that their visits would be as frequent as they are now, and just a few (20%) claim that because of the current information on the student's success their visits to school would be even more frequent (Figure 5).

Only some teachers (32%) would be willing to voluntarily participate in a test version of the programme. 29% would not participate, and 39% only if the management directed to.

In general, less than half of the teachers (43%) is in favour of an e-grade book. The result might be due to an age structure of the teachers.

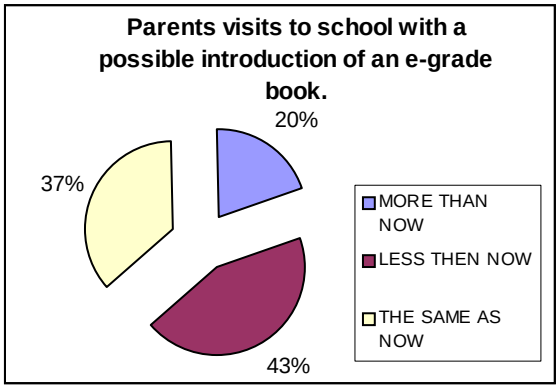


Figure 5: Teachers' opinions on parents' visits to school with a possible introduction of an e-grade book

4.2 Parents' Response

The majority of parents (67%) have already heard of an e-grade book, while the rest (33%) have never heard of it (Figure 6).

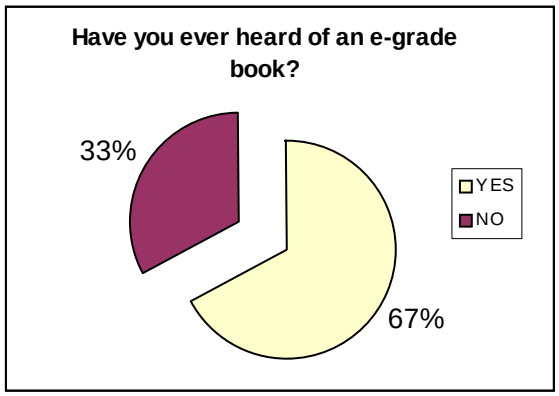


Figure 6: Parents' acquaintance with an e-grade book

Only a minority of parents (3%) are poorly familiar with their child's school success, the majority (53%) are well familiar with their success, some parents (44%) are extremely well familiar with their child's success.

Some parents (32%) get information on a child's results besides parents' meetings and parent-teacher interviews also extra, but majority (68%) do not.

Majority of parents is in favour of introducing an e-grade book (71%) some of them having doubts only about the security of information (see Figure 7). Minority is against (29%) and has the following opinions on the introduction of an e-grade book:

- It will encroach upon children's rights.
- We find an e-grade book unnecessary.
- 'Non-electronic' and 'non-internet' achievements and aids are more important for a child's success at school and in life.
- It would be a burden for a child.
- School success must not be dependent on the internet but on a child.
- I am interested in my child's grades and not my neighbour's child.
- In general I think the computer age has expanded too much for the children will not be able to read and talk.
- This leads to loneliness, which will soon become a big problem.

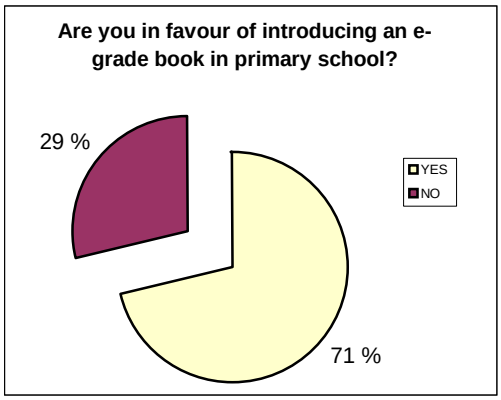


Figure 7: Parents' inclination towards introduction of an e-grade book

Most parents (85%) think that the success of their child after the introduction of an e-grade book would stay at the same level. Some (14%) think that it would be better or even a lot better, but there are also parents (1%) who think that the introduction of an e-grade book would deteriorate their child's success.

If monthly subscription for the use of an e-grade book was involved, the minority of parents (1%) would be prepared to pay 10 EUR, (9%) 5 EUR, (16%) less than 5 EUR. The majority (74%) thinks that the use should be free of charge.

We were also interested in the percentage of parents who have a computer either at home or at work, and if and how they are connected with the internet. The results were as follows: the majority of parents (84%) have a computer (at home or at work, Figure 8), out of these a little more than a half (51%) have a connection with the internet. Most of them have a connection with the internet over a cable connection (40%), a little less over on ordinary telephone line (32%), and 28% over a telephone line ISDN.

Parents with a computer and internet connection are absolutely in favour of introducing an e-grade book (50%). 31% agree with it but have some doubts about the information security, 15% don't agree and 4% are not interested in such form of a grade book (Figure 9).

23% of parents without a computer are absolutely in favour of introducing it, 41% are conditionally in favour since they have some doubts about the information

security. 16% are against the introduction of an e-grade book, and 20% of parents are not interested.

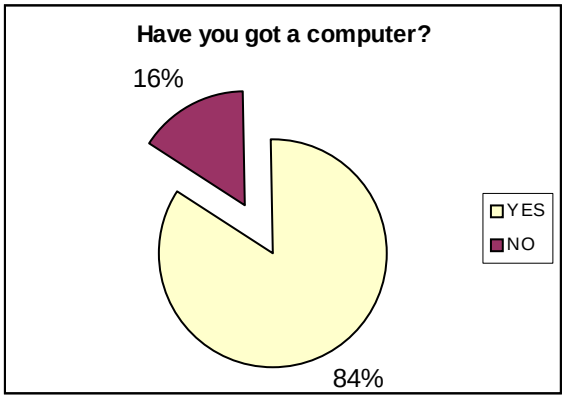


Figure 8: Parents equipment with computer either at home or at work

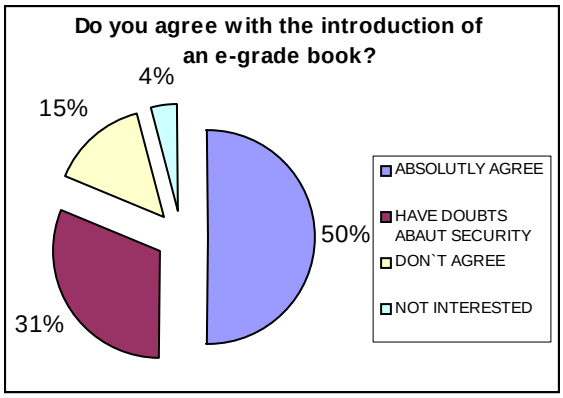


Figure 9: Parents' opinions that have a computer and internet connection about the introduction of an e-grade book

4.3 Responses Of Headmasters, Psychologists And Social Workers

The following conclusions have been reached after talking to other employees in primary schools.
In schools where the management is in favour of informational technology and follows its progress, the subordinates accept new technology and changes that follow its introduction much easier. Psychologists and social workers still have their own opinion about the new informational system (personal communication and personal relations between pupils – parents – teachers are more important), but a slight change in favour of introduction of an e-grade book is evident. This of course leads to easier and faster changes with the introduction of new business informational system (Jereb E. and Jereb J., 2000; Jereb, 2002).

The situation in schools where the management does not follow the informational technology and does not approve of the introduction of business informational system is just reverse. Here the management transfers its views to the subordinates who do not have many chances for success even if they agree or want changes about informational technology (Jereb E. and Jereb J, 2000). Such school are only few in numbers.

5 Discussion

Although the study only involved a small sample of potential users, it did demonstrate that the majority of parents having their children in primary schools are in favour of the introduction of an e-grade book. That is because parents will have access to their child's grades whenever they want to and children will not be able to keep their grades a secret. They will have access to information on knowledge checks, natural science days, cultural and sport days. And also access to teachers' notes to parents on their child's behaviour in class. The e-grade is also a good solution in a case parents cannot attend parent-teacher interviews or parents' meetings.

When the parents are not equipped with the knowledge or technology to access the before mentioned information, a problem may occur. It can also happen that due to an e-grade book parents will no longer have a direct contact with a class teacher and other teachers at school, which is also not so good.

A lot of teachers were also interested in introducing an e-grade book in primary schools. Since parents will be able to check their child's grades over the Internet and will be able to get all the necessary information that way, the teachers will not have to call parents home so often or visit them (due to bad grades, inexcusable absence, inappropriate behaviour).

Maybe the reason why more of them have not shown the interest is because teachers who are not familiar with the use of information communication technology will have to take more time to learn how to use the programme than those who are. Some of them might even dislike the new technology. It can also happen that they will not have a direct contact with all the parents any more. And teachers might need some extra time for registration of grades and other information.

Using an e-grade book instead of a classical form of a grade book enables an easier managing of the data since it is written in electronic form (can be printed in different forms, can be distributed over e-mail, can be transferred into other programmes, etc.). That also enables an easier statistical supervision.

What we are worried about is the security and protection of the system and a possible loss of data (human factor, machine failure).

6 Conclusion

Informatization in primary schools is expanding therefore an introduction of an e-grade book is to be expected in the near future. The technology of an e-grade

book offers the parents as well as teachers new possibilities. School management will have to consider the best possible way to use these possibilities and get with the progress. An e-grade book on the internet is an application which enables the parents to learn about their child's grades anytime and anywhere as long as they have connection with the internet as well as the password for their child's grades. An e-grade book must not be viewed as a substitution for parents' visits to school. We are still of the opinion that a personal contact is of the utmost importance. This application should serve as an addition for the parents to use in order to have some extra information on their child's grades, but the circumstances prevent them from doing so (either the class teacher cannot be reached or they cannot participate in parent-teacher interview). We believe the e-grade book will have a positive effect on a pupil-parent-teacher relation since the parents will have a current information on their child's grades which will also serve as a stimulation for their frequent visits to school for the interview with a class teacher.

References

[1] Adams, P.C., & Warf, B. (1997). Cyberspace and geographical space. *Geographical Review*, 87(2),139-145.

[2] Amundsen, M. (1999). *Using Visual InterDev*. Indianapolis: Que publishing.

[3] Cooper, B. (2000). Internet: Searching the Internet. London: Dorling Kindersley.

[4] Jereb, E., & Jereb, J. (2000). *Organizacija pisarniškega poslovanja*. Kranj: Moderna organizacija.

[5] Jereb, E. (2002). Avtomatizacija pisarniškega poslovanja. Kranj: Moderna organizacija.

[6] Mitchell, C.G., & Hope, B.G. (2002). Teaching or Technology: Who's Driving the Bandwagon? In E. Cohen, *Challenges of Information Technology in the 21st Century,* London: Idea Group Publishing.

[7] Papa, J., Caison, C., Debetta, P., & Wilson, E. (1999). *Professional ADO RDS programming with ASP.* Chicago: Wrox Press.

[8] Reisman, S. (1994). *Multimedia computing.* London: Idea group publishing.

[9] Schofield, S. (1994). *The UK Internet book.* England: Addison-Wesley.

[10] Sussman, D., & Homer, A. (1998). *Ado 2.0 - Programmer`s reference.* Chicago: Wrox Press.

[11] Šalamon, B. (1998). *Internet pojmovnik.*. Izola: Desk d.o.o.

Appendix A

QUESTIONNAIRE -PARENTS

Electronic grade book on the internet

Every grade the learner gets in the school is written in the e-grade book, just as by classic grading book. Only that hereby also the parents will have the insight into their child's grades, access to teacher's notes to the parents, their child's behavior etc. To help us introduce the advantages and problems which introducing of an e-grade book might bring alone, we ask you to fill in this questionnaire. Thank you.

1. **Have you ever heard of an e-grade book?**
   Yes No

2. **Are you familiar with your child's success in school?**
   a) poorly
   b) good
   c) very good
   d) do not know at all

3. **Would you be interested in constant overlook over your child's grades?**
   Yes No

4. **Besides attending regular parent-teacher interview do you ever extra ask about your child's success in school?**
   Yes No

5. **Are you interested in introducing an e-grade book, which enables to track your child's grades, behavior, teacher's notes, anytime you want to?**
   a) Yes, off course.
   b) Conditionally, I am worried about data security.
   c) No (why not?)_____
   d) Not interested in such an grade book.

6. **Do you think that introducing an e-grade book would change your child's success in school?**
   a) Yes to worse.
   b) It would not.
   c) Yes it would be better.
   d) I think it would be much better.

7. **Are you prepared to pay for using such a grade book?**
   a) No
   b) less than 5 EUR
   c) 5 EUR
   d) 10 EUR
   e) up to 20 EUR

8. **Do you have a computer (at home, at work)?**
   Yes No

9. If you do, are you connected to the internet?
Yes No

10. If yes, how is your connection?
a) cable TV
b) ISDN
c) classic telephone line

Appendix B

QUESTIONNAIRE -TEACHERS

Electronic grade book on the internet

Every grade the learner gets in the school is written in the e-grade book, just as by classic grading book. Only that hereby also the parents will have the insight into their child's grades, access to teacher's notes to the parents, their child's behavior etc. To help us introduce the advantages and problems which introducing of an e-grade book might bring alone, we ask you to fill in this questionnaire. Thank you.

1. What do you think of an e-grade book?
a.) Something new, I think it would be useful, worth trying.
b.) Only additional work for the teachers, waste of time. I am not interested.
c.) Do not know what you are asking about.
d.) Else_____

2. Your concern in case of introducing an e-grade book into primary schools.
a.) I am not concerned. I would just have to learn entering the grades.
b.) We would lose personal contact with parents.
c.) The e-grade system sounds to complicated for me.
d.) Else_____

3. Do you think parents would still come to school?
a.) More than now.
b.) Less.
c.) The same.

4. Would you collaborate in testing the pilot version of the e-grade system?
a.) Yes.
b.) No.
c.) Only if really necessary (management's directive).

Type Systems for Concurrent Programming Calculi

N. Raja and R.K. Shyamasundar
School of Technology & Computer Science
Tata Institute of Fundamental Research
Mumbai 400 005, INDIA
Email: {raja, shyam}@tifr.res.in

Keywords: Type theory, concurrency, process calculi

Received: March 12, 2003

We explore the role of types *in models of concurrent computation, particularly in the concrete setting of the asynchronous π-calculus. The major theme of this work may be summarized by the slogan* – "Wherever you see structure, think of *types*". *We propose* type *annotations not merely to channels, but also to the highly structured set of processes. The type system guarantees that well typed expressions cannot go wrong. Polymorphic process types formalize extant informal ideas regarding the channel passing and process passing approaches to process mobility. Further, subtyping relation between process types distinguishes between true concurrency and nondeterministic choice.*

1 Introduction

Type systems for sequential programming languages lead to many advantages [3, 20, 27]. In programming practice: types help in structuring programs, they assist in compile-time error detection, and they are useful in optimizing the target code during the compiling process. In the theoretical study of programming language concepts: types help in the creation of succinct metalanguages that act as models for the study of real-life programming languages, and they serve as intermediate code in the task of providing mathematical semantics for programming languages. All this has naturally led to investigations regarding the role of types in theories of concurrency.

The goal of this paper is to examine whether there are any benefits to be gained by introducing *types* in models for concurrent computation. In this paper, we illustrate the role of *types* in the concrete setting of the asynchronous π-calculus (API) [17, 6]. We choose API as it is one of the most prominent calculi for concurrency and communication. API has two kinds of entities – *names* (also called *channels*) and *processes* (also called *agents*). *Names* do not possess any structure, whereas a good amount of structure is needed to build *processes*. The type system we propose, assigns types to both processes and channels. The type assigned to channels, characterizes the length and the nature of the elements that the channel may carry in a communication. The type assigned to processes, characterizes the set of actions that the process is committed to. This results in a rich notion of types which is very useful in the monadic as well as the polyadic versions of API. The type system proposed shows that there are substantial benefits to be reaped by exploring the idea of typing processes. The usage of our type system entails the following advantages:

- It provides a scaffolding for the structured use of the π-calculus, by which we can abolish certain undesirable features – like infinite concurrent activity – right at the early stage of building process terms, rather than at the stage of the reduction system.
- Guarantees *safety*, that well typed expressions will not go wrong.
- Does not constrain the expressive power of the π-calculus.
- Our type system, with minor changes, can be applied to all process algebra formalisms of concurrency. Thus, it provides a uniform basis for the relative assessment of various formalisms. For example, polymorphism in process types brings out potential impredicativity in the semantics of some of these formalisms.
- Subtyping relation among process types helps in distinguishing true concurrency from nondeterministic choice.

The rest of this paper is organized as follows: Section 2 gives a brief review of the asynchronous π-calculus (API); Section 3 presents the type system; Section 4 shows that the type system preserves the semantics of API; Section 5 examines the type system with regard to those properties which are normally of interest in sequential languages; Section 6 explores further extensions to the type system – polymorphism and subtyping – by analogy with traditional type theory; Section 7 describes related work on concurrency and types. The conclusions and future research directions are presented in Section 8.

2 The Asynchronous π-Calculus

In this section, we include a brief review of the asynchronous π-calculus (API) [17, 6] notions that are required for this paper.

Following Milner's idea, a number of calculi for concurrent computation have been proposed, where the communication mechanisms are similar. Communication consists in synchronously sending and receiving through a shared labeled channel.

API [17, 6, 22, 25, 23, 35] is a model of concurrent computation that supports process mobility by naming and passing channels. It consciously forbids the transmission of processes as messages. One of its goals is to demonstrate that in some sense it is sufficiently powerful to allow only names to be the content of communications.

API has two kinds of entities – names (channels), and processes (agents).

Names $(x, y, \ldots \in \mathcal{X})$, have no structure.

Processes $(P, Q, \ldots \in \mathcal{P})$ possess a well defined structure given by

$P ::= 0 \mid \overline{x}y \mid x(y).P \mid P|Q \mid\ !P \mid (\nu x)P \mid$ ERROR

The construct $\overline{x}y$ outputs the name y along x, and does not bind y. The construct $x(y)$ inputs a name, say y, along x, and binds y in the prefixed process. The word 'asynchrony' in this calculus means that message output is non-blocking. This is ensured by restricting the formation of a term $\overline{x}y.P$ in the π-calculus to the case where P is an inactive process. API is powerful enough to encode the synchronous message passing discipline of the π-calculus [36, 30]. The term 0 represents an inactive process. We have extended the π-calculus by including a constant process called ERROR, to represent the kind of type mismatches that we wish to avoid at run-time. The form $P|Q$ means that P and Q are concurrently active, are independent, and can also communicate. The operator "!" is called replication, and $!P$ means $P|P|\ldots$; as many copies as you wish. Finally, $(\nu x)P$ restricts the use of *name* x to P. Apart from input prefix, "ν" is another mechanism for binding names within a process term in API. The operator "ν" may also be thought of as creating new channels.

The operational semantics of API is given in two stages, as shown in Figure 1. A structural congruence is first defined over the process terms, and then a reduction relation is defined. Notice that the rules do not allow reduction under prefix or replication. Also, as expected there are no reduction rules for ERROR. For more details about API, the reader is referred to [17, 6].

3 The Type System

We present our type system in three stages – first, the syntax; second, the typing rules corresponding to API process constructors; and finally, the typing rules corresponding to the reduction system of API. The following subsections are devoted to each of these three stages respectively. Though we use the monadic asynchronous π-calculus to illustrate our typing system, our results can be extended to the polyadic case in a straightforward manner.

3.1 Syntax for Types

We shall call the type information assigned to names as *sort* (ranged over by the metavariable s), and shall use the term *type* (ranged over by the metavariable t) to designate the type information assigned to processes. Our typing scheme is an implicit one (Curry-style typing), because we want to illustrate our work in the setting of a familiar calculus, without any syntactic modifications to the term structure of the calculus.

The sort 's' denotes the length and nature of names which a given channel may carry in a communication. The superscripts R, S, indicate that the channel usage as "receive mode" and "send mode" respectively.

In API, processes may be viewed as programs which manipulate names (which in turn can be considered as data). As mentioned earlier the data manipulated by API programs are unstructured entities. The data develops some structure only in the polyadic extension of API. In the monadic case, the data are atomic entities while in the polyadic case they are n-tuples. Thus the notion of sorts starts making sense only in the polyadic case.

On the other hand, processes have a well-formed structure even in the monadic case; hence types are of significance in both versions of API. The type 't' denotes a process type; it comes in various forms as depicted in Figure 2. The *arrow type* arises due to the *prefix* constructor; the *intersection type* arises due to *par*; and the *recursive type* arises due to *Bang*; the *internal* and *external* types arise due to the *hiding* operator. The API expressions leading to the above types will become clear as we look at each of the typing rules given in the following subsections.

3.2 Types for Processes

An API process $\alpha.A$ can be regarded as an action α and a continuation A. $\alpha.A$ is called a commitment – it is a process committed to act at α [22]. This is precisely the information that the type associated with a process embodies.

Proposition 3.1 (Process-types and Commitment) *A process type describes the sequence of actions that a process is committed to.*

This will become clear from the following subsections.

API is based on the object model of computing [26]. Objects have an independent identity and they have a persistent state which may not be entirely visible to the other agents. Thus the type associated with a process has two facets – one which specifies its interface on the outside and the other which determines its internal transitions. Our type system brings out this aspect of API explicitly by making

Definition 2.1 (Structural Congruence over Process Terms)
$\equiv$ is the smallest congruence relation over process terms such that the following laws hold:

1. Processes are identified if they only differ by a change of bound names
2. $(\mathcal{P}/\equiv, |, 0)$ is a symmetric monoid
3. $!P \equiv P|!P$
4. $(\nu x)0 \equiv 0, (\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$
5. If $x \notin freeNames(P)$ then $(\nu x)(P|Q) \equiv P|(\nu x)Q$
6. $P|\text{ERROR} \equiv !\text{ERROR} \equiv (\nu x)\text{ERROR} \equiv \text{ERROR}$

Definition 2.2 (Reduction Relation)
The reduction relation $\rightarrow$ over processes is the smallest relation satisfying the following rules:

$$\text{Comm} \quad (\ldots + x(y).P) \mid (\ldots + \overline{x}[z].0) \rightarrow P\{y \leftarrow z\} \mid 0$$

$$\text{Par} \quad \frac{P \rightarrow P'}{(P|Q) \rightarrow (P'|Q)}$$

$$\text{Struct} \quad \frac{Q \equiv P \quad P \rightarrow P' \quad P' \equiv Q'}{Q \rightarrow Q'}$$

$$\text{Res} \quad \frac{P \rightarrow P'}{(\nu x)P \rightarrow (\nu x)P'}$$

Figure 1: Operational Semantics of API

Sorts	$s ::= BasicSort \mid (s)^R \mid (s)^S$
Type − Variables	$T \mid U \mid V$
Pre − Types	$\sigma ::= \epsilon \mid \phi \mid T \mid Name(Name : s)^R \mid Name(Name : s)^S \mid \sigma \rightarrow \sigma \mid \sigma \cap \sigma \mid \mu T.\sigma$
Pre − Types	$\sigma \mid \sigma_{ext} \mid \sigma_{int}$
Types	$t ::= < \sigma_{ext}, \sigma_{int} >$
TypeEnvironments	$\Gamma ::= \{\} \mid \Gamma, x : s \mid \Gamma, P : t$

Figure 2: Syntax of the Type System

Zero	$\Gamma \vdash 0 : \phi$
Prefix − R	$\frac{\Gamma \vdash x{:}(s)^R,\ y{:}s \quad \Gamma \vdash P{:}t}{\Gamma \vdash x(y).P : x(y{:}s)^R \rightarrow t}$
Prefix − S	$\frac{\Gamma \vdash x{:}(s)^S,\ y{:}s}{\Gamma \vdash \overline{x}(y) : x(y{:}s)^S}$
Par − I	$\frac{\Gamma \vdash P{:}t_1 \quad \Gamma \vdash Q{:}t_2}{\Gamma \vdash P \mid Q : t_1 \cap t_2}$
Bang	$\frac{\Gamma \vdash P{:}(t_1 \rightarrow t_2)}{\Gamma \vdash\ !P : \mu T.t_1 \rightarrow (t_2 \cap T)}$
New − Channel	$\frac{\Gamma \vdash P{:}t \quad \Gamma \vdash x{:}s}{\Gamma \vdash (\nu x)P{:}<t[x \leftarrow \epsilon],\ t>}$

Figure 3: Typing Rules for Process Constructors

the type associated with a process to be a tuple comprising its external and internal types respectively.

The typing rules corresponding to each of the process constructors that API allows, are listed in Figure 3. Among all the typing rules listed in Figure 3, the internal and external types turn out to be distinct only when the 'hiding operator' occurs in the process term. Hence, only the *New-Channel* typing rule shows both components of the *type* associated with a process term. The types are to be viewed as being implicitly universally quantified on name sorts. The typing rules are given in a syntax directed way, and can be checked for well-formedness by structural induction over the API syntax.

Arrow types

Arrow types are familiar from type systems for sequential programming. The typing rule *Prefix-R* states in its premises that if x, y are names, y has *sort* s, and x has sort $(s)^R$ – which means that the channel x may be used for receiving a *name* of sort s – and the process P has type t; then the API term $x(y).P$ is assigned the type $x(y : s)^R \rightarrow t$. The type indicates that process $x(y).P$ can use channel x for receiving only, indicated by the superscript R. Further, after such a communication occurs (and only after), it may proceed to behave like a process having type t. This strict sequentiality imposed by the prefix constructor of API is made explicit by the $\rightarrow$. The rule *Prefix-S* is very similar except that it shows that the name x may be used only for sending (the superscript S) by the newly constructed process.

We shall discuss the prefix rule again when we consider higher-order models for concurrency. The *Prefix* type rules will reveal any impredicativity which could be lurking in the semantics of the calculus being typed. More about impredicativity will be discussed in Section 7.

Intersection types

The rule *Par-I* says that the *intersection type* '$\cap$' arises when a process is built by the parallel composition of two other process terms. The parallel composition operator '|' allows the components to make transitions independently (i.e., disjoint parallelism). Thus, the set of actions that a process belonging to an intersection type can indulge in, is given by the conjunction of the set of possible actions of its component processes. Intersection types are also called 'conjunctive types' in the parlance of type theory.

There is a notable difference between the conventional usage of intersection types [3], and the way they are used in this work. In this work, the intersection type corresponds to a process constructor (par, '|'). Traditionally, intersection types are used for typing a term which belongs to various structurally unrelated types. For example, the symbol '+' is used to represent integer addition, and real addition. The type assigned to such a function is $((int \rightarrow int \rightarrow int) \cap (real \rightarrow real \rightarrow real))$. In other words, conventional intersection types are used to represent 'overloading'. Notably also absent from our type system, is the universal type ω (such that $P : \omega$ for all terms P), which accompanies intersection types normally.

The parallel composition operator '|' also allows the

components to communicate. Hence we shall encounter the *intersection type* '$\cap$' once again in the typing rule describing communication between the two component processes.

Recursive Types

The *Bang* typing rule is another instance where the relevance of types in concurrency is very clearly brought out. The operator "!" is called replication and $!P$ – "bang P"– means $P|P\ldots$; as many copies as you wish. In API the "!" operator can be applied to any process term P to form the process $!P$ (where P has been constructed using any rule for building processes). The important point to be noted is that API does not enforce any restrictions on P before the "!" operator may be applied to it.

However the typing rule *Bang* states in its premise that the type of P should be an "arrow type" such as $(t_1 \rightarrow t_2)$ before we can apply "!" to P to get $!P$. This makes it mandatory that the outermost constructor of process P be a prefix, before the "!" may be applied to it. Thus the replication operator can be used on guarded processes only. $!\pi.P$ is a common instance of replication – it indicates a resource P which can be replicated only when a requester communicates via π. This shows that the premise in the typing rule *Bang* is meaningful. The next question which arises is whether the typing rule *Bang* is being too restrictive by imposing such a condition. Before we answer this query, let us examine the meaning of a term such as $!P$ when it is not required of P that its outermost constructor be a prefix. Such a term, "$!P$", means a resource which replicates asynchronously – replicates without demand, without requirement. $!P$ appears to be acting on its own *free will*, so to say. In other words it represents *infinite concurrent activity*. Now this is certainly not a meaningful construct, and we would rather not have such a term in our calculus. Hence the typing rule *Bang* does not strip API off any expressive power; in fact it rules out an entire class of meaningless terms from being constructed. API abolishes such behaviour by taking recourse to its reduction rules. However we have done better in our type system, in that, we even forbid the occurrence of such terms right at the level of syntax, by enforcing a discipline in the structured construction of API programs.

After having looked at the premise, let us now examine the conclusion of the *Bang* rule. It infers that the process term $!P$ has the type $(\mu T.t_1 \rightarrow (t_2 \cap T))$. μ represents recursion and the type variable T is the parameter of the recursion. The recursive type makes the recursive behavior of "!" operator explicit. The intuition provided by the recursive type is well supported when we turn to API and find that all parametric recursive process definitions can be encoded by replication. Let us come back to the *Bang* typing rule: When P has the type $(t_1 \rightarrow t_2)$, it means that P behaves as dictated by the type t_1 and then (sequentially) behaves as dictated by t_2. The recursive type assigned to $!P$ says that $!P$ behaves as required by t_1 and then as required by $(t_2 \cap T)$. The intersection type mirrors the fact that an independent process of type t_2 has been spawned, which executes in parallel with the resource of type T. But T is the parameter of recursion, and we eliminate it by recursive unfolding, that is we replace T by $(\mu T.t_1 \rightarrow (t_2 \cap T))$ and proceed further as before.

The Recursive type in this setting is very similar to that used in sequential programming. The type $\mu T.t_p$ stands for the least fixed point solution of the type equation $T = t_p$. The solutions of such equations will be infinite types, which can be represented by infinite labeled binary trees. The definition of such trees is provided in Figure 4. The same Figure also gives a congruence relation on types with the help of such trees [9, 10].

Internal and External Types

In all the typing rules that we have considered so far, the external and internal types are identical. However, the operator ν used as $(\nu x)P$ localizes (restricts) the use of the channel x within P. The channel name x is guaranteed to be different from any other channel name which finds an occurrence outside P. Hence communications can be sent and received on x only internally within process P. This brings us to the next typing rule, *New-Channel*, which gives the external and internal type of a process term which has been built using the operator ν. The notion of distinguishing between the external and internal type of a process is derived from the notion of existential types and explicit witnesses [28], and the notion of partially abstract types [8]. The *external* type-component states that if the process P has type t and the channel x has sort s, then the external type of the process term $(\nu x)P$ is $t[x \leftarrow \epsilon]$ which means that in the type t all occurrences of x are replaced by ϵ, thereby making the channel x unavailable for communication with the outside world. The *internal* type-component states that as far as the internal type of $(\nu x)P$ is concerned, there is no change, the type continues to be t.

That explains all the typing rules that have arisen because of the process constructors that are allowed in API.

3.3 Reduction rules and Types

The typing rules shown in Figure 4 correspond to the congruence relation over types. They spell out when two types may be considered to be congruent.

The remaining typing rules, shown in Figure 5, correspond to the reduction system of API. The typing rules *Inter-E, Comm, Par-R, Res*, and *Struct* tell us how to consistently infer the type of the term which results from a reduction.

The rule *Comm* mentions the types required of each term so that the communication between the two processes will result in a proper reduction (one which does not result in ERROR), and gives the type of the resultant process. The rule *Par-I* mentioned earlier as giving rise to the *intersection type* '$\cap$', can be considered to be a special case of this rule. If there is no communication possibility allowed by

Definition 3.2
The tree corresponding to the process type t, written as $T(t)$, is defined as follows:

$T(\phi) = \phi$;

$T(t_1 \rightarrow t_2) = (\rightarrow, T(t_1), T(t_2))$;

$T(t_1 \cap t_2 = (\cap, T(t_1), T(t_2))$;

$T(\mu T.\, t) = T(t[T \leftarrow \mu T.\, t])$.

Definition 3.3
$\approx_t$ is the smallest congruence relation over types, such that, the following laws hold:

CR-1 Process types are identified if they only differ by a change of bound names;

CR-2 $t \cap \phi \approx_t \phi \cap t \approx_t t$;

CR-3 $t_1 \approx_t t_2$, if $T(t_1) = T(t_2)$.

Figure 4: Congruence Relation for Types

$$\text{Inter} - \text{E} \quad \frac{\Gamma \vdash P{:}t_1 \cap t_2}{\Gamma \vdash P{:}t_1 \qquad \Gamma \vdash P{:}t_2}$$

$$\text{Comm} \quad \frac{x(y).P : (x(y{:}s)^R \rightarrow t_P), \quad \overline{x}(z) : (x(z{:}s)^S)}{x(y).P \mid \overline{x}(z) \rightarrow P\{y \leftarrow z\} : t_P\{y \leftarrow z\}}$$

$$\text{Par} - \text{R} \quad \frac{P{:}t_P \rightarrow P'{:}t_{P'}}{(P|Q) : t_P \cap t_Q \rightarrow (P'|Q) : t_{P'} \cap t_Q}$$

$$\text{Res} \quad \frac{P{:}<t_P, t_P> \rightarrow P'{:}<t_{P'}, t_{P'}>}{(\nu x)P{:}<t_P[x \leftarrow \epsilon], t_P> \rightarrow (\nu x)P'{:}<t_{P'}[x \leftarrow \epsilon], t_{P'}>}$$

$$\text{Struct} \quad \frac{t_Q \equiv_t t_P \quad P{:}t_P \rightarrow P'{:}t_{P'} \quad t_{P'} \equiv_t t_{Q'}}{Q{:}t_Q \rightarrow Q'{:}t_{Q'}}$$

Figure 5: Reduction Rules and Types

the types of the interacting processes (disjoint parallelism), then the resulting type of the compound term is given by the *Par-I* typing rule. It is worth noting that the typing rules corresponding to process constructors and the typing rules corresponding to the reduction system, cannot be kept separated in the type system for API. This is because the operator *par* '|' is overloaded – it represents both concurrency (a process building operation), as well as communication (an operation which is a part of the reduction rules). However, such a clear separation can be achieved in the case of a type system constructed along similar lines for Boudol's *concurrent λ-calculus* [5].

Once again we mention that in all these rules, except *Res* the inference is valid for both components of the process type – external as well as internal. The *Res* typing rule explicitly indicates the process type as a tuple and gives the corresponding new components of the type after reduction.

4 Soundness and Type Safety

In this section, we examine the effect of the type-system on the semantics of API. First, we show that our type system preserves the semantics of API, and prove that well typed expressions never reduce to ERROR – which means process types guarantee the safety property.

The operational semantics of API was defined in two stages [22, 26] as shown in Section 2. A structural equivalence on process terms was given first, and then a reduction relation was given which describes the act of communication. We prove below that our notion of *type* is consistent with each of these two stages.

Theorem 4.1 *Types preserve the structural congruence rules on process terms.*

Proof: We prove this theorem by examining the structure of the definition of structural congruence on process terms.

1. Types respect α-conversion (typing rule *CR-1*), hence agents are identified if they only differ by a change of bound names.

2. Using the typing scheme presented in this paper, we show that types preserve the fact that $(\mathcal{P}/\equiv, |, 0)$ is a symmetric monoid.
   $$0 : \phi \quad (Zero)$$
   $$P|0 : t \cap \phi \quad (Par-I)$$
   $$t \cap \phi \approx_t t \quad (CR-2)$$
   Similarly,
   $$0|P : \phi \cap t \quad (Par-I)$$
   $$\phi \cap t \approx_t t \quad (CR-2)$$
   By steps 3 and 5, it follows that types preserve the monoidal structure of $P/\equiv$, where '|' is the associative operator of the monoid, and 0 forms the identity w.r.t '|'.

3. The typing rule *Bang* has been explained in sufficient detail in Section 3. It clearly follows from the illustration given there that types guarantee $!P \equiv P|!P$.

4. The inactive process 0 has the type ϕ as both its external and internal type. The restricted process $(\nu x)0$ continues to have the same type. Hence $(\nu x)0 \equiv 0$.
   If the process P has the process type $< E_P, I_P >$ then the process term $(\nu x)(\nu y)P$ has the type $< E_P[x \leftarrow \epsilon, y \leftarrow \epsilon], I_P >$ which is equivalent to the type $< E_P[y \leftarrow \epsilon, x \leftarrow \epsilon], I_P >$ associated with the process term $(\nu y)(\nu x)P$.

5. From the typing rules *Par-I*, and *New-Channel* it immediately follows that if x is not free in P then $(\nu x)(P|Q) \equiv P|(\nu x)Q$.

Thus *types* preserve the structural congruence on process terms. □

Theorem 4.2 *Well typed expressions can never reduce to* ERROR.

Proof: In the absence of types, the reduction rule which allows communication between process terms states that $x(y).P \mid \overline{x}z \rightarrow P\{y \leftarrow z\}$. The typing scheme assigns to each of the two concurrent process terms the following types –
$x(y).P : x(y : s)^R \rightarrow t_P$, and $\overline{x}(z) : x(z : s)^S$
Further the type scheme allows a reduction to take place by the typing rule *Comm* only when the two types are complementary and the sorts of the channels being used for communication are consistent with each other. These are exactly the conditions required to ensure a meaningful reduction in the π-calculus. The term resulting from the communication is $P\{y \leftarrow z\}$ and its corresponding type is $t_P[y \leftarrow z]$. Then well typed process terms never reduce to ERROR. □

Theorem 4.3 *The type system preserves the semantics of* API.

Proof: Follows as a direct consequence of Theorem 4.1 and Theorem 4.2. □

5 Basic Syntactic Properties

The type system proposed in this work is meant for concurrent calculi, and as is well known, the requirements of concurrent systems are quite different from those of sequential systems. However, there are a number of syntactic properties which have been of interest in traditional type systems for sequential programming [3]. For the sake of completeness we briefly examine such properties in our type system.

1. **Implicit Typing:** The typing scheme we have proposed is an implicit one (Curry-style typing). We chose Curry-style typing because we wanted to illustrate our work in the setting of a familiar calculus without requiring major syntactic modifications to the term structure of the calculus.

2. **Church-Rosser Property (CR):** This is more a property of the underlying calculus being typed, rather

than the type system itself. In our case, API does not satisfy the Church-Rosser property, since functions such as 'parallel-or' can be represented in it.

3. **Subject Reduction (SR):** If process term P has the type t_P, and if P reduces to the term P'; then the subject reduction property states that the type of P' is also t_P. Such a property does not hold in our type system because process reduction in API is non-deterministic, and also due to name passing, the interface of a process may change with reduction.

4. **Strong Normalization (SN):** This property states that all reduction sequences terminate eventually. This means that not every computable function is definable in the system. However this property does not hold in our type system because of the presence of recursive process types. With the help of recursive process types we are able to type the "!" operator of API without restricting its expressive power.

5. **Type Checking:** This property states whether, given a typing environment Γ, a process term P, and a type t, is the judgment $\Gamma \vdash P : t$ decidable or not. Type checking is decidable for our type system.

6. **Type Inference:** This requires that given Γ and P, it should be possible to compute a t such that $\Gamma \vdash P : t$ is valid. Type inference is possible for process types.

The above properties gained prominence because of their importance in the traditional application areas of types, such as in proof theory and in sequential programming. In the domain of concurrency, many of the above properties such as CR, SR, and SN are no longer relevant. Instead, properties such as *safety* and *liveness* become important.

6 Further Extensions to the Type System

There are a number of concepts which have played a significant role in the success of type disciplines for sequential systems. Two such concepts are *Polymorphism* and *Subtyping*. In this section we examine whether these concepts shed any light on concurrent calculi. We informally extend our type discipline in two directions – to incorporate polymorphism and subtyping. The results are indeed very promising as we demonstrate in the following subsections. Further research along these lives is sure to lead to insights into concurrent calculi.

6.1 Channel passing versus Process passing

Many distinct formalisms [25, 29, 37, 1, 18, 2, 5] have been invented to describe systems which do not have fixed interconnection topology between processes. All such formulations may be classified into two groups by examining the way in which they achieve mobility. One group achieves mobility by allowing channel names to be communicated [25, 1, 18] – the π-calculus belongs to this group. The other group achieves mobility by supporting the transmission of processes as messages [37, 2, 29, 5] – let us take a particular example from this group, say CHOCS [37].

The name passing approaches to concurrency allow names, but not processes, to be transmitted in communications. On the other hand, the process passing approaches allow processes, but not names, to be transmitted as messages. There are relevant reasons why each of these two approaches allows only either names or processes but not both to be the content of communications. Thus neither of the two approaches can be said to have achieved "uniformity" in dealing with their primitive entities. Further it has been demonstrated [37, 22, 36] that both the paradigms are equally powerful as far as their expressive power is concerned.

The question that we ask now is whether our type system can provide any relevant criteria that favours the choice of one paradigm over the other? The answer is in the affirmative – the type system does provide a measure which helps in discriminating the two paradigms.

In order to see how, let us examine the type that our system assigns to the process constructor which allows abstraction of names and processes in the paradigms of name-passing and process-passing calculi respectively. In this section, let x, y range over *Names*; P, Q range over *Processes*; N_s range over *Name Sorts*; P_t and t_q range over *Process Types*.

Consider the following π-calculus term, and its corresponding type – $x(y).Q : \forall N_s.x(y : N_s)^R \rightarrow t_q$. The type expression states that the process term $x(y).Q$ behaves like a program which expects any name y as input (y is a dummy parameter), and then behaves like the process Q. However there is no restriction on what sort of name it can accept as input, as shown by the universal quantifier which ranges over N_s. The important point to be observed is that the entity "$\forall N_s.x(y : N_s)^R \rightarrow t_q$" is itself a process type and does not lie in the range of the universal quantifier (which ranges only over name sorts in this case).

Now consider the following CHOCS term, and its corresponding type – $x?(P).Q : \forall P_t.x(P : P_t)^R \rightarrow t_Q$. In this case the type expression states that the process term $x?(P).Q$ behaves like a program which expects any P process as input (P is a dummy parameter), and then behaves like the process Q. However the program does not impose any restrictions on the type of the input process (represented by the universal quantifier ranging over P_t. In this case the entity "$\forall P_t.x(P : P_t)^R \rightarrow t_Q$" is itself a process type and hence the universal quantifier ranges over this type as well. In other words process types turn out to be impredicative in CHOCS, while they remain predicative in the π-calculus.

It is a well known phenomenon in type theory that the semantics of a predicative formalism is extremely simple and elegant in comparison with the semantics required by an impredicative formalism [11]. Thus conceptual simplicity

and elegance in the semantics of the type system associated with a formalism favours π-calculus over CHOCS – or in more general terms, name passing approaches over process passing approaches to concurrency.

6.2 True Concurrency versus Nondeterministic Interleaving

As mentioned in Section 7, the work by Pierce and Sangiorgi has shown that the subtyping relation among name sorts leads to an interesting refinement. In this subsection we examine the relevance of subtyping relation among process types.

In the semantic theories for process algebras such as CCS [21] and CSP [14], concurrency is semantically reduced to nondeterminism. For example the process $a|b$ is considered semantically equivalent to the process $(a.b + b.a)$. It has been demonstrated by Boudol et al. [7], that in certain situations it is meaningful to retain concurrency as a primitive concept without reducing it to nondeterministic interleaving. We now show that process types can be used to maintain such a distinction.

For this purpose we introduce *union types*, '$\cup$', and a *subtyping* relation among union types. Consider a process term of the form $P + Q$. This term can (nondeterministically) indulge, either in the actions specified by P or in the actions specified by Q (exclusive-or of the actions). If the types of P and Q are given by t_p and t_q respectively, then we assign to the process $P + Q$, the type $t_p \cup t_q$. Now we define the subtyping relation '$\subseteq$', by the relations, $t_p \subseteq (t_p \cup t_q)$ and $t_q \subseteq (t_p \cup t_q)$. The *subtyping* relation is reflexive, antisymmetric, and transitive. Intuitively in a context which requires an object of type t, one could as well use an object whose type is a subtype of t, but not vice versa. This intuition is well supported when we examine the process terms themselves. It is important to note that such a subtyping relationship does not hold in the case of intersection types i.e. $t_p \not\subseteq (t_p \cap tq)$ and $t_q \not\subseteq (t_p \cap t_q)$.

Thus we get the type of $a|b$ as $(t_a \cap t_b)$ and the type of $((a.b) + (b.a))$ as $((t_a \rightarrow t_b) \cup (t_b \rightarrow t_a))$. Consider the above processes after they make a transition on 'a'. $(a.b) + (b.a)$ reduces to b. The new process type is a subtype of the original process type, i.e. $t_b \subseteq ((t_a \rightarrow t_b) \cup (t_b \rightarrow t_a))$. On the other hand the process $a|b$ also reduces to b. But the distinction lies in the fact that the new process type is not a subtype of the original process type, i.e. $t_b \not\subseteq (t_a \cap t_b)$. Thus the type equivalence provided by the subtype relation provides a key to distinguish true concurrency from nondeterministic interleaving.

7 Related Work

In this section we briefly discuss work related to type systems for mobile processes. As mentioned earlier, the concurrent calculi that were proposed following Milner's CCS, have two basic syntactic entities – *channels* and *processes*. This situation is unlike that in sequential programming, where the λ-calculus (the de-facto standard sequential language), has only one basic entity – *terms*. Till now a major part of the research on type systems for concurrency has concentrated on assigning type information to the channels only. Such type information has been called *sorts*.

The relevant starting point is the notion of *sorts* introduced in the polyadic π-calculus by Milner [22]. We illustrate Milner's notion of *sorting* with an example. Consider the process term $\overline{x}y.0|x(u).\overline{u}().0|\overline{x}z.0$. In this expression, channels y and z carry only the empty vector if they are ever used for communication. On the other hand, channel x always carries another channel name, which in turn is used in communicating an empty vector. We can represent these observations as: $\{y \mapsto (), z \mapsto (), x \mapsto (())\}$. Notice that the usage of x is characterized by a nesting of parentheses. The above representation is precisely the *sorting* as proposed by Milner. Thus the *sort* associated with a channel captures the length and nature of the vector that the name carries in communications. In the polyadic π-calculus, names may carry n-tuples of other names. Hence the notion of sort information assumes prominence only in the polyadic setting. There are some more points to be noted. Firstly, sort information is assigned to channels only and sort equivalence is by *name matching*. Secondly, names occurring in a perfectly meaningful π-calculus process term may not have any *sorting* at all. This can occur if a term uses names to communicate different entities at different times. Thus the lack of a proper *sorting* does not render a π-calculus expression meaningless. Finally, *sorts* are implicit i.e., they do not occur in the term structure of the calculus. Honda [15] presented similar results, in an independent work. Gay [12] presents an algorithm (quadratic in the length of the input process) for automatically inferring such sort information for channels, from the given π-calculus term. Naturally *sorts* are inferred only if they exist. Honda and Vasconcelos [16] gave an algorithm to the same effect, though linear in the size of the input process. Following Lafont's work on interaction nets [19], Honda proposed conditions on channel sorts, so as to achieve freedom from deadlock in certain finite and simple situations.

Pierce and Sangiorgi [31] extended the notion of sorts by distinguishing between the ability to read from a channel, the ability to write to a channel, and the ability to do both. This refinement gives rise to an interesting subtype relation on channel sorts. Their sort equivalence is by *structural matching*. In Pierce's work, sorts appear explicitly in the term structure and further such sort information is even communicated from one process to another. This requires changes in the π-calculus model, thus resulting in a different concurrent calculus. In Pierce's work, the problem of algorithmic inference of sort information is not considered at all.

The idea of assigning type information to processes has also been used by researchers in other contexts [29, 13, 34]. In Facile, CML, and the Typed λ-calculus with first class processes, the notion of process type is present. However

the process types which find usage in these programming languages are predominantly just functional types. The notion of polymorphism has been included in Facile and CML, but once again in the realm of channel sorts, in order to derive more flexible sorting mechanisms.

From the above observations it is clear that the notions of type inference, polymorphism, subtyping, and conditions for deadlock freedom have been explored in the domain of channel sorts. Such investigations in the domain of process types, would yield rich dividends [33, 4, 32, 38, 39].

8 Conclusions and Future Directions

The aim of this work was to establish a bridge between the disciplines of concurrency and type theory. We presented a novel operational semantics for the asynchronous π-calculus, by making reductions sensitive to type. Our type system was unique, in not confining type information to channels only; very informative types were assigned to processes also. The universe of process terms with its rich structure, proved to be a fertile ground for the application of various type constructors. The type system did not restrict the expressive power of the asynchronous π-calculus in any way. Types guaranteed *safety*, that well typed expressions would not go wrong. Further the type system helped in preventing the construction of meaningless expressions, such as those representing infinite concurrent activity, right at the stage of syntactic formation of process terms. The notion of polymorphism brought out the latent impredicativity in the semantics of the process-passing approaches to concurrency. The notion of subtyping helped in distinguishing true concurrency from nondeterministic interleaving.

As further work, it would be highly interesting and relevant to explore how the notion of process types could be put to use in reasoning about *liveness* properties of concurrent systems, such as freedom from deadlock. It would also be fruitful to pursue work towards establishing algebraic equivalences over process types. Also as discussed in the last section, exploring the notions of polymorphism and subtyping looks promising.

This work is part of an ongoing investigation into the role of type theoretic concepts in the setting of concurrency. It would also be productive to carry out such an investigation in a more abstract formalism for concurrency, e.g., like the one provided by *action structures* [24].

Acknowledgment

We wish to thank the anonymous referees for constructive comments which were of help in improving the content and presentation of this paper. Our thanks to Ms. Margaret D'Souza for typing and typesetting this paper.

References

[1] E. Astesiano, and G. Reggio (1984) Parametric Channels via Label Expressions in CCS, *Theor. Comp. Science*, Vol. 33, pp. 45–64.

[2] E. Astesiano and G. Reggio (1987) SMoLCS-driven concurrent calculi, *Lecture Notes in Computer Science*, Springer-Verlag, Vol. 249, pp. 169–201.

[3] H. Barendregt, and K. Hemerik (1990) Types in lambda calculi and programming languages, *Proc. ESOP'90*, LNCS 432, pp. 1–36.

[4] M. Berger, K. Honda, and N. Yoshida (2003) Genericity and the π-Calculus, *Proc. FOSSACS'03*, LNCS, To appear.

[5] G. Boudol (1989) Towards a lambda-calculus for concurrent and communicating systems, *Proc. TAPSOFT'89*, LNCS 351, Springer-Verlag, pp. 149–161.

[6] G. Boudol (1992) Asynchrony and the π-calculus, *Rapport de Recherche*, Number 1702, INRIA Sophia-Antipolis.

[7] G. Boudol, I. Castellani, M. Hennessy, and A. Kiehn (1991) Observing Localities, *INRIA Report No. 1485*.

[8] L. Cardelli, and P. Wegner (1985) Understanding Types, Data Abstraction, and Polymorphism, *ACM Computing Surveys*, Vol. 17 (4).

[9] F. Cardone, and M. Coppo (1990) Two Extensions of Curry's Type Inference System, *Logic and Computer Science*, Academic Press, pp. 19–75.

[10] B. Courcelle (1983) Fundamental Properties of Infinite Trees, *Theoretical Computer Science*, Vol. 25, pp. 95–169.

[11] R.L. Constable (1991) Type Theory as a Foundation for Computer Science, *Proc. TACS'91*, Lecture Notes in Computer Science, Vol. 526, Springer-Verlag.

[12] S. Gay (1993) A sort inference algorithm for the polyadic π-calculus, *Proc. ACM Symposium on Principles of Programming Languages*, ACM Press.

[13] A. Giacolone, P. Mishra, and S. Prasad (1989) Facile: A symmetric integration of concurrent and functional programming, *Int. Jl. of Parallel Prog.*, Vol. 18, pp. 121–160.

[14] C.A.R. Hoare (1985) Communicating Sequential Processes, *Prentice-Hall*, London.

[15] K. Honda (1993) Types for Dyadic Interaction, *Proc. CONCUR'93*, Lecture Notes in Computer Science, Volume 715, Springer-Verlag.

[16] K. Honda, and V.T. Vasconcelos (1993) Principal typing schemes in a polyadic π-calculus, *Proc. CONCUR'93*, LNCS 715, Springer-Verlag.

[17] K. Honda, and M. Tokoro (1991) An Object Calculus for Asynchronous Communication, *ECOOP'91*, Lecture Notes in Computer Science, Volume 512, Springer-Verlag.

[18] S.R. Kennaway and M.R. Sleep (1985) Syntax and informal semantics of DyNe, a parallel language, *LNCS 207*, Springer-Verlag, pp. 222–230.

[19] Y. Lafont (1990) Interaction Nets, *Proc. POPL'90*, ACM Press, pp. 95–108.

[20] B. Mahr (1993) Applications of Type theory, *Proc. TAPSOFT'93*, Lecture Notes in Computer Science, Volume 668, Springer-Verlag.

[21] R. Milner (1989) Communication and Concurrency, *International Series in Computer Science*, Prentice Hall.

[22] R. Milner (1991) The polyadic π-calculus: a tutorial, *Logic and Algebra of Specification*, Proceedings of International NATO Summer School (Marktoberdorf, Germany), Series F, Vol. 94, Springer.

[23] R. Milner (1999) Communicating and Mobile Systems: The Pi Calculus, Cambridge University Press.

[24] R. Milner (1993) Action Structures and the π-Calculus, *Proof and Computation*, Proceedings of International NATO Summer School (Marktoberdorf, Germany), Series F, Vol. 139, Springer.

[25] R. Milner, J. Parrow, and D. Walker (1992) A calculus of mobile processes (Parts I and II), *Information and Computation*, Vol. 100, pp. 1–77.

[26] R. Milner (1992) Functions as processes, *Journal of Mathematical Structures in Computer Science*, Vol. 2 (2), pp. 119–141.

[27] J.C. Mitchell (1990) Type Systems for Programming Languages, *Handbook of Theoretical Computer Science*, Elsevier Science Publishers.

[28] J.C. Mitchell, and G. Plotkin (1988) Abstract Types have Existential Types, *ACM Transactions on Programming Languages and Systems*, Vol. 10 (3).

[29] F. Nielson (1989) The typed λ-calculus with first class processes, *Proc. PARLE'89*, Lecture Notes in Computer Science, Volume 366, Springer-Verlag.

[30] C. Palamidessi (1997) Comparing the expressive power of the Synchronous and the Asynchronous pi-calculus, *Proc. ACM Symposium on Principles of Programming Languages*, ACM Press, pp. 256–265

[31] B. Pierce, and D. Sangiorgi (1993) Typing and Subtyping for Mobile Processes, *Proc. IEEE Symposium on Logic in Computer Science*, IEEE Press.

[32] B. Pierce, and D. Sangiorgi (2000) Behavioral Equivalence in the Polymorphic Pi-Calculus, *Proc. Journal of ACM*, Vol. 47 (3) pp 531–584.

[33] N. Raja, and R.K. Shyamasundar (1994) Type Systems for Concurrent Calculi, *Proc. of the Tenth Workshop on Abstract Data Types* (ADT'94), Santa Margherita Ligure, Genoa, Italy.

[34] J.H. Reppy (1993) Concurrent ML: Design, Application and Semantics, *Funct. Prog., Concurrency, Simulation and Automated Reasoning*, LNCS 693, Springer-Verlag.

[35] D. Sangiorgi, and D. Walker (2001) The Pi-Calculus – A Theory of Mobile Processes, Cambridge University Press.

[36] D. Sangiorgi (1993) From π-calculus to Higher-Order π-calculus — and back, *Proc. TAPSOFT '93*, Lecture Notes in Computer Science, Volume 668, Springer-Verlag.

[37] B. Thomsen (1993) Plain CHOCS. A Second Generation Calculus for Higher Order Processes, *Acta Informatica*, Vol. 30 (1), pp. 1–59.

[38] D.N. Turner (1996) The Polymorphic Pi-Calculus: Theory and Implementation, *Ph.D. Thesis*, University of Edinburgh.

[39] V. Vasconcelos (1994) Typed Concurrent Objects, *Proc. ECOOP'94*, LNCS, Springer-Verlag, pp. 100–117.

JOŽEF STEFAN INSTITUTE

Jožef Stefan (1835-1893) was one of the most prominent physicists of the 19th century. Born to Slovene parents, he obtained his Ph.D. at Vienna University, where he was later Director of the Physics Institute, Vice-President of the Vienna Academy of Sciences and a member of several scientific institutions in Europe. Stefan explored many areas in hydrodynamics, optics, acoustics, electricity, magnetism and the kinetic theory of gases. Among other things, he originated the law that the total radiation from a black body is proportional to the 4th power of its absolute temperature, known as the Stefan–Boltzmann law.

The Jožef Stefan Institute (JSI) is the leading independent scientific research institution in Slovenia, covering a broad spectrum of fundamental and applied research in the fields of physics, chemistry and biochemistry, electronics and information science, nuclear science technology, energy research and environmental science.

The Jožef Stefan Institute (JSI) is a research organisation for pure and applied research in the natural sciences and technology. Both are closely interconnected in research departments composed of different task teams. Emphasis in basic research is given to the development and education of young scientists, while applied research and development serve for the transfer of advanced knowledge, contributing to the development of the national economy and society in general.

At present the Institute, with a total of about 700 staff, has 500 researchers, about 250 of whom are postgraduates, over 200 of whom have doctorates (Ph.D.), and around 150 of whom have permanent professorships or temporary teaching assignments at the Universities.

In view of its activities and status, the JSI plays the role of a national institute, complementing the role of the universities and bridging the gap between basic science and applications.

Research at the JSI includes the following major fields: physics; chemistry; electronics, informatics and computer sciences; biochemistry; ecology; reactor technology; applied mathematics. Most of the activities are more or less closely connected to information sciences, in particular computer sciences, artificial intelligence, language and speech technologies, computer-aided design, computer architectures, biocybernetics and robotics, computer automation and control, professional electronics, digital communications and networks, and applied mathematics.

The Institute is located in Ljubljana, the capital of the independent state of S**love**nia (or S♡nia). The capital today is considered a crossroad between East, West and Mediterranean Europe, offering excellent productive capabilities and solid business opportunities, with strong international connections. Ljubljana is connected to important centers such as Prague, Budapest, Vienna, Zagreb, Milan, Rome, Monaco, Nice, Bern and Munich, all within a radius of 600 km.

In the last year on the site of the Jožef Stefan Institute, the Technology park "Ljubljana" has been proposed as part of the national strategy for technological development to foster synergies between research and industry, to promote joint ventures between university bodies, research institutes and innovative industry, to act as an incubator for high-tech initiatives and to accelerate the development cycle of innovative products.

At the present time, part of the Institute is being reorganized into several high-tech units supported by and connected within the Technology park at the Jožef Stefan Institute, established as the beginning of a regional Technology park "Ljubljana". The project is being developed at a particularly historical moment, characterized by the process of state reorganisation, privatisation and private initiative. The national Technology Park will take the form of a shareholding company and will host an independent venture-capital institution.

The promoters and operational entities of the project are the Republic of Slovenia, Ministry of Science and Technology and the Jožef Stefan Institute. The framework of the operation also includes the University of Ljubljana, the National Institute of Chemistry, the Institute for Electronics and Vacuum Technology and the Institute for Materials and Construction Research among others. In addition, the project is supported by the Ministry of Economic Relations and Development, the National Chamber of Economy and the City of Ljubljana.

Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
Tel.:+386 1 4773 900, Fax.:+386 1 219 385
Tlx.:31 296 JOSTIN SI
WWW: http://www.ijs.si
E-mail: matjaz.gams@ijs.si
Contact person for the Park: Iztok Lesjak, M.Sc.
Public relations: Natalija Polenec

INFORMATICA

AN INTERNATIONAL JOURNAL OF COMPUTING AND INFORMATICS

INVITATION, COOPERATION

Submissions and Refereeing

Please submit three copies of the manuscript with good copies of the figures and photographs to one of the editors from the Editorial Board or to the Contact Person. At least two referees outside the author's country will examine it, and they are invited to make as many remarks as possible directly on the manuscript, from typing errors to global philosophical disagreements. The chosen editor will send the author copies with remarks. If the paper is accepted, the editor will also send copies to the Contact Person. The Executive Board will inform the author that the paper has been accepted, in which case it will be published within one year of receipt of e-mails with the text in Informatica LaTeX format and figures in `.eps` format. The original figures can also be sent on separate sheets. Style and examples of papers can be obtained by e-mail from the Contact Person or from FTP or WWW (see the last page of Informatica).

Opinions, news, calls for conferences, calls for papers, etc. should be sent directly to the Contact Person.

QUESTIONNAIRE

☐ Send Informatica free of charge

☐ Yes, we subscribe

Please, complete the order form and send it to Dr. Drago Torkar, Informatica, Institut Jožef Stefan, Jamova 39, 1111 Ljubljana, Slovenia.

Since 1977, Informatica has been a major Slovenian scientific journal of computing and informatics, including telecommunications, automation and other related areas. In its 16th year (more than ten years ago) it became truly international, although it still remains connected to Central Europe. The basic aim of Informatica is to impose intellectual values (science, engineering) in a distributed organisation.

Informatica is a journal primarily covering the European computer science and informatics community - scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the Refereeing Board.

Informatica is free of charge for major scientific, educational and governmental institutions. Others should subscribe (see the last page of Informatica).

ORDER FORM – INFORMATICA

Name: .....

Title and Profession (optional): .....

.....

Home Address and Telephone (optional): .....

.....

Office Address and Telephone (optional): .....

.....

E-mail Address (optional): .....

Signature and Date: .....

Informatica WWW:

http://ai.ijs.si/informatica/
http://orca.st.usm.edu/informatica/

Referees:

Witold Abramowicz, David Abramson, Adel Adi, Kenneth Aizawa, Suad Alagić, Mohamad Alam, Dia Ali, Alan Aliu, Richard Amoroso, John Anderson, Hans-Jurgen Appelrath, Iván Araujo, Vladimir Bajič, Michel Barbeau, Grzegorz Bartoszewicz, Catriel Beeri, Daniel Beech, Fevzi Belli, Simon Beloglavec, Sondes Bennasri, Francesco Bergadano, Istvan Berkeley, Azer Bestavros, Andraž Bežek, Balaji Bharadwaj, Ralph Bisland, Jacek Blazewicz, Laszlo Boeszoermenyi, Damjan Bojadžijev, Jeff Bone, Ivan Bratko, Pavel Brazdil, Bostjan Brumen, Jerzy Brzezinski, Marian Bubak, Davide Bugali, Troy Bull, Sabin Corneliu Buraga, Leslie Burkholder, Frada Burstein, Wojciech Buszkowski, Rajkumar Bvyya, Giacomo Cabri, Netiva Caftori, Particia Carando, Robert Cattral, Jason Ceddia, Ryszard Choras, Wojciech Cellary, Wojciech Chybowski, Andrzej Ciepielewski, Vic Ciesielski, Mel Ó Cinnéide, David Cliff, Maria Cobb, Jean-Pierre Corriveau, Travis Craig, Noel Craske, Matthew Crocker, Tadeusz Czachorski, Milan Češka, Honghua Dai, Bart de Decker, Deborah Dent, Andrej Dobnikar, Sait Dogru, Peter Dolog, Georg Dorfner, Ludoslaw Drelichowski, Matija Drobnič, Maciej Drozdowski, Marek Druzdzel, Marjan Družovec, Jozo Dujmović, Pavol Ďuriš, Amnon Eden, Johann Eder, Hesham El-Rewini, Darrell Ferguson, Warren Fergusson, David Flater, Pierre Flener, Wojciech Fliegner, Vladimir A. Fomichov, Terrence Forgarty, Hans Fraaije, Stan Franklin, Violetta Galant, Hugo de Garis, Eugeniusz Gatnar, Grant Gayed, James Geller, Michael Georgiopolus, Michael Gertz, Jan Goliński, Janusz Gorski, Georg Gottlob, David Green, Herbert Groiss, Jozsef Gyorkos, Marten Haglind, Abdelwahab Hamou-Lhadj, Inman Harvey, Jaak Henno, Marjan Hericko, Henry Hexmoor, Elke Hochmueller, Jack Hodges, Doug Howe, Rod Howell, Tomáš Hruška, Don Huch, Simone Fischer-Huebner, Zbigniew Huzar, Alexey Ippa, Hannu Jaakkola, Sushil Jajodia, Ryszard Jakubowski, Piotr Jedrzejowicz, A. Milton Jenkins, Eric Johnson, Polina Jordanova, Djani Juričič, Marko Juvancic, Sabhash Kak, Li-Shan Kang, Ivan Kapustøk, Orlando Karam, Roland Kaschek, Jacek Kierzenka, Jan Kniat, Stavros Kokkotos, Fabio Kon, Kevin Korb, Gilad Koren, Andrej Krajnc, Henryk Krawczyk, Ben Kroese, Zbyszko Krolikowski, Benjamin Kuipers, Matjaž Kukar, Aarre Laakso, Sofiane Labidi, Les Labuschagne, Ivan Lah, Phil Laplante, Bud Lawson, Herbert Leitold, Ulrike Leopold-Wildburger, Timothy C. Lethbridge, Joseph Y-T. Leung, Barry Levine, Xuefeng Li, Alexander Linkevich, Raymond Lister, Doug Locke, Peter Lockeman, Vincenzo Loia, Matija Lokar, Jason Lowder, Kim Teng Lua, Ann Macintosh, Bernardo Magnini, Andrzej Małachowski, Peter Marcer, Andrzej Marciniak, Witold Marciszewski, Vladimir Marik, Jacek Martinek, Tomasz Maruszewski, Florian Matthes, Daniel Memmi, Timothy Menzies, Dieter Merkl, Zbigniew Michalewicz, Armin R. Mikler, Gautam Mitra, Roland Mittermeir, Madhav Moganti, Reinhard Moller, Tadeusz Morzy, Daniel Mossé, John Mueller, Jari Multisilta, Hari Narayanan, Jerzy Nawrocki, Rance Necaise, Elzbieta Niedzielska, Marian Niedq'zwiedziński, Jaroslav Nieplocha, Oscar Nierstrasz, Roumen Nikolov, Mark Nissen, Jerzy Nogieć, Stefano Nolfi, Franc Novak, Antoni Nowakowski, Adam Nowicki, Tadeusz Nowicki, Daniel Olejar, Hubert Österle, Wojciech Olejniczak, Jerzy Olszewski, Cherry Owen, Mieczyslaw Owoc, Tadeusz Pankowski, Jens Penberg, William C. Perkins, Warren Persons, Mitja Peruš, Fred Petry, Stephen Pike, Niki Pissinou, Aleksander Pivk, Ullin Place, Peter Planinšec, Gabika Polčicová, Gustav Pomberger, James Pomykalski, Tomas E. Potok, Dimithu Prasanna, Gary Preckshot, Dejan Rakovič, Cveta Razdevšek Pučko, Ke Qiu, Michael Quinn, Gerald Quirchmayer, Vojislav D. Radonjic, Luc de Raedt, Ewaryst Rafajlowicz, Sita Ramakrishnan, Kai Rannenberg, Wolf Rauch, Peter Rechenberg, Felix Redmill, James Edward Ries, David Robertson, Marko Robnik, Colette Rolland, Wilhelm Rossak, Ingrid Russel, A.S.M. Sajeev, Kimmo Salmenjoki, Pierangela Samarati, Bo Sanden, P. G. Sarang, Vivek Sarin, Iztok Savnik, Ichiro Satoh, Walter Schempp, Wolfgang Schreiner, Guenter Schmidt, Heinz Schmidt, Dennis Sewer, Zhongzhi Shi, Mária Smolárová, Carine Souveyet, William Spears, Hartmut Stadtler, Stanislaw Stanek, Olivero Stock, Janusz Stokłosa, Przemysław Stpiczyński, Andrej Stritar, Maciej Stroinski, Leon Strous, Ron Sun, Tomasz Szmuc, Zdzislaw Szyjewski, Jure Šilc, Metod Škarja, Jiři Šlechta, Chew Lim Tan, Zahir Tari, Jurij Tasič, Gheorge Tecuci, Piotr Teczynski, Stephanie Teufel, Ken Tindell, A Min Tjoa, Drago Torkar, Vladimir Tosic, Wieslaw Traczyk, Denis Trček, Roman Trobec, Marek Tudruj, Andrej Ule, Amjad Umar, Andrzej Urbanski, Marko Uršič, Tadeusz Usowicz, Romana Vajde Horvat, Elisabeth Valentine, Kanonkluk Vanapipat, Alexander P. Vazhenin, Jan Verschuren, Zygmunt Vetulani, Olivier de Vel, Didier Vojtisek, Valentino Vranić, Jozef Vyskoc, Eugene Wallingford, Matthew Warren, John Weckert, Michael Weiss, Tatjana Welzer, Lee White, Gerhard Widmer, Stefan Wrobel, Stanislaw Wrycza, Tatyana Yakhno, Janusz Zalewski, Damir Zazula, Yanchun Zhang, Ales Zivkovic, Zonling Zhou, Robert Zorc, Anton P. Železnikar

Informatica

An International Journal of Computing and Informatics

Archive of abstracts may be accessed at USA: http://, Europe: http://ai.ijs.si/informatica, Asia: http://www.comp.nus.edu.sg/ liuh/Informatica/index.html.

Subscription Information Informatica (ISSN 0350-5596) is published four times a year in Spring, Summer, Autumn, and Winter (4 issues per year) by the Slovene Society Informatika, Vožarski pot 12, 1000 Ljubljana, Slovenia.
The subscription rate for 2004 (Volume 28) is
– USD 80 for institutions,
– USD 40 for individuals, and
– USD 20 for students
Claims for missing issues will be honored free of charge within six months after the publication date of the issue.

LaTeX Tech. Support: Borut Žnidar, Kranj, Slovenia.
Lectorship: Fergus F. Smith, AMIDAS d.o.o., Cankarjevo nabrežje 11, Ljubljana, Slovenia.
Printed by Biro M, d.o.o., Žibertova 1, 1000 Ljubljana, Slovenia.

Orders for subscription may be placed by telephone or fax using any major credit card. Please call Mr. R. Murn, Jožef Stefan Institute: Tel (+386) 1 4773 900, Fax (+386) 1 219 385, or send checks or VISA card number or use the bank account number 900–27620–5159/4 Nova Ljubljanska Banka d.d. Slovenia (LB 50101-678-51841 for domestic subscribers only).

Informatica is published in cooperation with the following societies (and contact persons):
Robotics Society of Slovenia (Jadran Lenarčič)
Slovene Society for Pattern Recognition (Franjo Pernuš)
Slovenian Artificial Intelligence Society; Cognitive Science Society (Matjaž Gams)
Slovenian Society of Mathematicians, Physicists and Astronomers (Bojan Mohar)
Automatic Control Society of Slovenia (Borut Zupančič)
Slovenian Association of Technical and Natural Sciences / Engineering Academy of Slovenia (Igor Grabec)
ACM Slovenia (Dunja Mladenič)

Informatica is surveyed by: AI and Robotic Abstracts, AI References, ACM Computing Surveys, ACM Digital Library, Applied Science & Techn. Index, COMPENDEX*PLUS, Computer ASAP, Computer Literature Index, Cur. Cont. & Comp. & Math. Sear., Current Mathematical Publications, Cybernetica Newsletter, DBLP Computer Science Bibliography, Engineering Index, INSPEC, Linguistics and Language Behaviour Abstracts, Mathematical Reviews, MathSci, Sociological Abstracts, Uncover, Zentralblatt für Mathematik

The issuing of the Informatica journal is financially supported by the Ministry of Education, Science and Sport, Trg OF 13, 1000 Ljubljana, Slovenia.

Volume 28 Number 1 April 2004 **ISSN 0350-5596**

Informatica

An International Journal of Computing and Informatics

Informatica **28** (2004) Number 1, pp. 1–115