Agent Interoperation Across Multagent System Boundaries

Joseph A. Giampapa
The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
garof@cs.cmu.edu

Massimo Paolucci
The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
paolucci@cs.cmu.edu

Katia Sycara
The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
katia@cs.cmu.edu

ABSTRACT

Recently the number of autonomous agents and multiagent systems (MAS) that have been developed by different developers has increased. Despite efforts for the creation of standards (eg. in communication languages, registration protocols etc.), it is clear that at least in the near term heterogeneous agents and MASs will be prevalent. Therefore, mechanisms that allow agents and/or MASs to interoperate and transact are needed. In this paper we report on a case study and lessons learned of an interoperator agent we developed. We discuss requirements for interoperation mechanisms, resulting challenges and our design decisions and implementation of the RETSINA-OAA InterOperator[1].

1. INTRODUCTION

One of the problems facing open, multiagent systems [MAS] operating on the Internet is that as the number of MAS architecture-specific agent communities increases, so too does the difficulty of locating and collaborating with agents in communities of different MAS architectures. Each MAS has its own architecture-specific features such as: agent registration, agent capability advertisements, strategy for finding agents, agent communication language [ACL], agent dialogue mediation, default agent query preference, and agent content language, to name a few. Since MASs are open, that is, architectures that allow agents to dynamically enter and exit agent communities without any restrictions on the number or types of participants, there is the further constraint that whatever the solution, it must act in real-time so as to capture the dynamism of the agent world. If an agent enters one agent community, for example, agents in another community should know about that event, just as they should know if it eventually leaves so that they will not count on its services any more. All the above features present many different challenges with respect to agent and MAS interoperations. Despite the many efforts for standardizing various aspects of agent-based systems, we do not believe that homogeneity can be achieved with respect to the above MAS architecture-specific features. Therefore, interoperation mechanisms must be designed and implemented.

Solutions to MAS interoperability must be chosen carefully so as to have little negative impact on the interoperated systems. Not making the interoperator transparent, for example, means that all agents of a community must be retrofitted with a new protocol for conversing with it, and this negatively influences the overall effectiveness of the interoperator to support scalability.

In this paper, we describe the issues and design challenges regarding the design and implementation of an interoperator. We define a multiagent system interoperator as an entity that provides agents of one MAS architecture access to the desired capabilities and services offered by another MAS architecture. We describe a case study of interoperating between the RETSINA [7] capability-based MAS architecture and SRI's Open Agent Architecture [OAA] [5], where the differences between the two systems are described by the above characterizations.

The RETSINA-OAA InterOperator acts as a connection between the RETSINA system and the OAA system. The task of the InterOperator is to allow any agent in the RETSINA system to access any service or information provided by OAA agents, and for any agent in the OAA system to access services or information provided by RETSINA agents.

Agents in a multiagent system should "speak the same language" to understand each other. Agents in the OAA system "speak" Prolog-based OAA ICL, while agents in the RETSINA system use KQML [3]. As discussed at length below, the difference between these two languages is not just a matter of superficial rewriting, rather they are characterized by very different syntactic and semantic structures. Because the agents cannot understand each other's languages, they cannot communicate without some sort of a translation system.

The difference between the two systems is not only on the language level, but at the architectural level, as well. OAA is organized around an agent called the *Facilitator*, which manages all the communications between agents, in such a

[1]The authors thank David Martin, Adam Cheyer and Didier Guzzoni at SRI for their support and help. Marcus Huber of OGI was instrumental in explaining the operation of the OAA-KQML Bridge. In addition we are especially grateful to Martin Van Velsen for his help with the graphical display of the RETSINA and OAA systems and with the data collection for the evaluation section. This research has been sponsored in part by the Office of Naval Research Grant N-00014-96-16-1-1222 by DARPA grant F-30602-98-2-0138.

way that no two OAA agents directly communicate with each other. On the other hand, the RETSINA system is constructed on the principle that all agents in the MAS community should communicate with each other. Agents in the RETSINA system find each other through a Matchmaker agent, but, in contrast to the Facilitator, the Matchmaker does not manage the transaction between agents. The Matchmaker allows agents to find each other and then allows them to interact with each other directly. Because of these different communication protocols any agent that enters the other system will address its messages to the wrong receiver. For instance, upon entering the RETSINA system, an OAA agent that needs stock quotes might send a request for a service to the Matchmaker expecting the Matchmaker to act like a Facilitator and to contact the provider. The Matchmaker would return "Secapl", the name of one of RETSINA's stock-reporting agents, as its return value. The OAA agent, expecting the return value to its query, might then interpret the Matchmaker's response as the actual value of a stock quote.

In conclusion, no agent that has been designed for one of the two systems can correctly interact with any of the agents designed for the other system due to differences in MAS agent communication languages, architectures, and the protocols of agent communication modes. The task of the RETSINA-OAA InterOperator is to overcome these limitations.

The paper is organized in two parts. In the first part, we describe the principles that guided the implementation of the RETSINA-OAA InterOperator, how the characteristics of the RETSINA and OAA MASs affected its design, and how the design has been implemented. In the second part, we describe the language differences between the two systems and how the RETSINA-OAA InterOperator translates messages from one system to the other. We conclude with an example of interoperation between the two systems and an evaluation of its performance.

2. MULTIAGENT INTEROPERATION

The five principles that guided our design of the RETSINA-OAA InterOperator are explained below.

1. MAS interoperators should maintain distinct MAS architecture boundaries. There are inevitable architecture-determined influences on the ways in which agents communicate with each other within the same agent community. If an interoperator requires that agents of one system acquire the ACL of the other architecture then the end result will be a merged-architecture ACL. This ACL acquisition process is an open problem to do automatically if the two systems must be modified by hand; it is an expensive process.

2. MAS interoperators should be scalable in order to preserve the open systems architectures of both participating MASs. One of the strengths of MASs is that they enable the dynamic arrival and departure of agents at runtime so as to promote continual system capability enhancement as much as limit the rapidity of overall system degradation. If MASs can only support a specific or fixed number of agents with a limited variety of protocols, then the desired level of dynamism of the combined agent architectures is not obtained. In the worst case, it would be more cost-effective to directly wrap a foreign agent in a native architecture's communication wrapper.

3. MAS interoperators should present an increase in savings relative to the amount of effort that must be invested in their development, so as to have increased functionality for the agent system. As a way of developing a rough initial metric, consider the cost of enhancing an agent community by wrapping "foreign" agents in the ACL communications API of the target MAS. The cost-to-increased-capability ratio can be said to be $1:1$ — add API:receive new agent. Proceeding in this way, the cost of wrapping M agents in one MAS's communications API, and the cost of wrapping N agents in another MAS's communications API will be $M+N$ for the increased capabilities of $M+N$ new agents in the combined MAS systems. Now consider the amount of effort to develop an interoperator for the two systems. We could simplistically assume that this incurs a constant cost, C: the costs of adding and maintaining two communications APIs, and the costs of translating advertisements, queries and other infrastructure ACL content for each MAS. If the sum $M+N$ is more than C, then there is already an advantage in the cost savings of adding new agents via MAS interoperability than by agent wrapping.

4. MAS interoperators should cross register agent capabilities from one MAS architecture community to another so as to maintain maximum accessibility of both systems to each other's capabilities. Furthermore, this cross registration should be performed in real time so as to maintain a high fidelity representation of agent community state.

5. MAS Interoperational Transparency. Agents should dialogue with each other across MAS boundaries without being aware that the interoperator is present. This eliminates the need to develop additional agent-interoperator protocols that could limit the applicability of an interoperator to different agent dialogue contexts. And without the cost of retro-fitting existing agents with new protocols, there is the added benefit that MAS interoperability promotes the scalability of agent systems: the more agents that are added to a MAS, the lower the unit cost of bringing them into the system and maintaining them.

3. DESIGN CONSIDERATIONS

The RETSINA-OAA InterOperator "bridges" the two worlds of RETSINA and OAA by advertising RETSINA agents with the OAA Facilitator, advertising OAA agents with the RETSINA Matchmaker, and by enabling agents from both RETSINA and OAA worlds to send messages to each other. As noted earlier, there are some architecture-specific features that are particular to each MAS and have some impact on the design of an interoperator. We shall consider those which applied in particular to our system.

3.1 MAS Community Membership

The need for MAS interoperability to maintain distinct MAS boundaries, and to provide transparent interoperability between agents of different systems, determined the fact that the RETSINA-OAA InterOperator is designed to belong to both systems. As a RETSINA agent, the RETSINA-OAA InterOperator can interact with all the other RETSINA agents in the system using the same communication protocols and KQML as communication language. Symmetrically, as an OAA agent, the RETSINA-OAA InterOperator can communicate to the OAA Facilitator using Prolog clauses.

3.2 Capability Advertising

Both RETSINA and OAA MASs are capability-based, that is two agents communicate with each other based on the ability of one agent to respond to the needs of the other. But the two systems are very different in the ways that they allow their agents to identify each other. The RETSINA MAS is matchmaker-based [1]: the RETSINA Matchmaker responds to a request for agents possessing certain capabilities by sending back a list of agent names and possibly also rankings of how well those agents suit the requester's needs. The requester is then free to contact any agent in that list according to its own personal criteria, which may include whether or not the provider agent is part of the same MAS. This approach is robust in that the requester is independent of the Matchmaker once it has the list of candidate agents to contact. The OAA system, on the other hand, employs facilitated matching to both find provider agents and to also coordinate agent communication within the OAA system. This approach is optimized for reflecting changes in the status of agents in the community.

A way of ensuring the design principles of interoperator transparency in a capability-based multiagent system is for it to advertise the capabilities of the agents from the other MAS. One benefit is that agents of each system can continue to participate in the agent selection process the way they normally do without following a new selection policy. Another benefit is that the InterOperator, itself, need not support the capability matching strategies particular to each system.

A question that is frequently raised when describing the RETSINA-OAA InterOperator is whether or not the names of the agents should be associated with their corresponding advertisements in the other agent world, or if just the name of the InterOperator should be associated with those capabilities. We coined the term *ProxyAgent* to describe the InterOperator if it supports distinct agent name identities in the other MAS, and *SuperAgent* if the InterOperator represents all the capabilities of one MAS to the other as if they were its own. It is possible for an interoperator to act as both types — one for each MAS community.

The RETSINA-OAA InterOperator presents itself as a SuperAgent to both the RETSINA and OAA agent communities. The primary reason for this was the expediency of managing a single communication thread for each agent community rather than implement multithreaded communications, as would be required for a ProxyAgent.

4. INTEROPERATOR ARCHITECTURE

The architecture of the RETSINA-OAA InterOperator is illustrated by figure 1. The InterOperator was constructed on the concept of three levels of abstraction. Starting with the bottom level, the layers are: the MAS-specific API layer, the MAS-specific agent session layer, and the aptly named interoperability layer.

The MAS-specific API layer is the software layer that is offered by each multiagent system for the development of agents in that particular architecture. It is represented here as a foundational layer that is responsible for passing and receiving messages to and from the MAS-specific agent session layer. The RETSINA system offers its Communicator, the package that offers peer-to-peer agent communications in KQML, ANS-based registry, log facility configuration, and

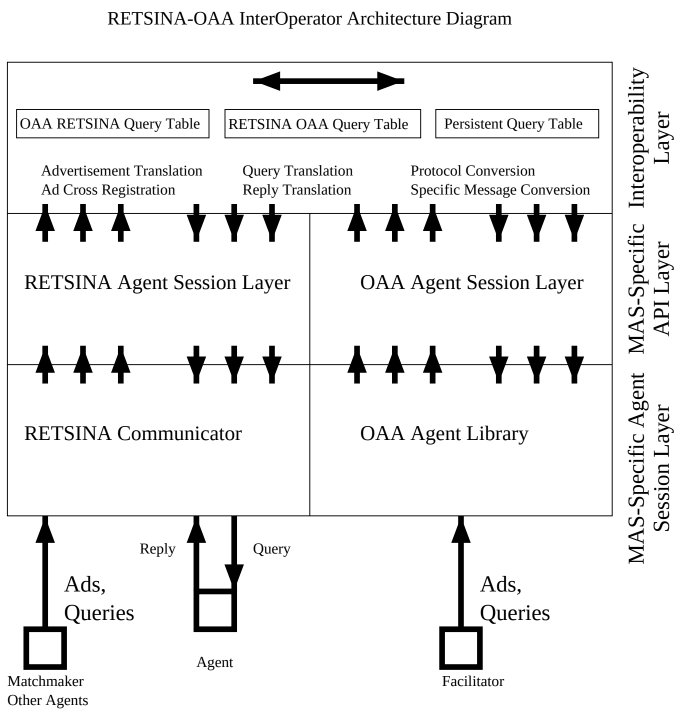

Figure 1: RETSINA-OAA InterOperator Architecture Diagram

KQML message and protocol construction and parsing utilities. The OAA system offers its Agent Library which contains libraries for posting advertisements to the Facilitator, sending Facilitator-mediate messages to other agents, posting agent queries, methods for accessing Facilitator-specific data structures, and utilities for composing and parsing OAA ICL messages. There is no interface between the RETSINA and OAA APIs at this level.

As represented in the diagram, agents establish communication connections with the InterOperator via this layer. On the RETSINA side, the InterOperator maintains one unique connection to which it listens for receiving advertisements, queries and for sending the translated OAA agent replies to RETSINA agent queries. If an OAA agent wishes to query a RETSINA agent, then the InterOperator initiates a connection and dialogues with the RETSINA agent on behalf of the OAA agent. On the OAA side, the InterOperator maintains only one connection with the Facilitator. Through that connection it both transmits and receives advertisements, replies, and other messages, indifferent of the mode of agent communication that is being supported by that connection.

The MAS-specific agent session layer is where agent communication sessions are logically maintained and agent communication protocols enforced. For example, if agents must send acknowledgement messages to the InterOperator, this is the level where the software “waits” for the reply and executes contingency methods should the acknowledgement not return. The specific protocols that are maintained for the RETSINA side of the InterOperator are those for communicating with the Matchmaker, for replying to RETSINA agent queries, and for initiating queries to RETSINA agents on behalf of OAA agents. Similarly on the OAA side, the protocols that are maintained there are specific for sending and receiving advertisements, queries, and other message types to and from the Facilitator. As for the API layer, interoperability does not occur at this level, either. The processes at this level communicate with the processes at the levels above or below them.

The top level of the InterOperator architecture is the interoperability layer. This is the layer where advertisements, messages, and query/response protocols are translated from one architectural convention to the other. This layer contains three data tables which are interesting for the needs they fulfill. These are the: 1. OAA-RETSINA Query Table [ORQT], 2. RETSINA-OAA Query Table [ROQT], and the 3. Persistent Query Table [PQT].

The purposes of the first two tables, the OAA-RETSINA Query Table and the RETSINA-OAA Query Table, are symmetric. Both MASs require that when there is a reply to an information agent query, that reply must contain the form of the original query in addition to the response. There is also the symmetry that both systems need to maintain a query thread, so that multiple agents of one system can simultaneously query the same agent of the other MAS, or so that one agent can launch multiple queries to the same agent of the other architecture while waiting for its replies. The third table, the Persistent Query Table, is a variant of the ROQT, except that it uses a different Facilitator data structure for effecting it. All three tables allow an association between a unique thread ID, the content of the original query, and the persistency mode (that which determines over how much time replies will be received) of the query.

The existence of these tables implies that the InterOperator actively maintains the "state" of the transactions occurring between the agents of the two interoperating systems. Maintaining the state of the agent dialogues is less robust than if the InterOperator did not maintain any state at all. Should the InterOperator fail and then come back on-line it is a non-trivial task to discover which queries have already terminated and which ones are on-going. One way of overcoming this is to pass the state information, such as message thread and original query message format, as a tag-along parameter to the server agent, as is already done for some queries to the OAA Facilitator. But the drawback of this approach is that the state information significantly increases the size of the query message, and message size has a direct negative impact on network throughput.

5. MESSAGE TRANSLATION

The main function of the RETSINA-OAA InterOperator is to support the transparent transmission of messages between RETSINA and OAA agents. As part of this process the RETSINA-OAA InterOperator needs to translate messages between the KQML language used by RETSINA agents and the Prolog style language used by OAA.

Agents in a Multiagent System should "speak the same language" to be able to understand each other's messages. Therefore, the specification of the language used by the agents is one of the parts of the definition of a MAS.

RETSINA uses KQML as its communication language. Following this format, each message has the form:

(performative :key1 value1 :key2 value2)

Performative is a predefined speech act like *tell* or *ask-one* that specifies the type of communicative action that the agent performs. For example *tell* is used to transfer information, while *ask-one* is used to send a question with the understanding that the other part will send an answer. *Key1* and *key2* are indexes that distinguish between different values in the message without any predefined agreement on their order. Finally, values are either atomic like a string or an integer, or a more complicated form in KQML format.

OAA's messages have the format of logic predicates following the Prolog format:

solvable(Goal, Parameters, Permissions)

The functor *solvable* of the predicate specifies what action should be performed as a consequence of the message. In this sense, the solvables' role is equivalent to the performative in KQML. Specifically, solvable is used to advertise the capabilities of the agent. The arguments of the predicate are used to specify the content of the message. In this case *Goal* reports what queries the agent should solve, while *Parameters* and *Permissions* specify additional constraints.

Prolog and KQML language formats are not simple syntactic variations of each other. For example, mapping the performative, the keys and values of the KQML form onto the functor and the arguments of the predicate creates a host of problems that are reviewed below.

Arguments vs Keys Arguments in Prolog predicates have a well defined and fixed order. In the predicate solvable shown above, the goal should always be in first position, parameters in second and permissions in the third. KQML does not make such an assumption, rather, values are associated with keys, and keys can be in any place in the form. Thus, the two following KQML forms are equivalent (perf :key1 value1 :key2 value2) and (perf :key2 value2 :key1 value1), while perf(value1, value2) and perf(value2, value1) are not. Any mapping of values in a KQML form into arguments in a parameter need a specification of the relation between keys and argument positions. This mapping information is provided neither by the KQML form, nor by the Prolog predicate.

Message Interpretation OAA and RETSINA interpret their messages in two radically different ways: OAA uses a predicative representation, while RETSINA uses a functional representation. OAA agents exchange Prolog predicates. Each agent tries to "prove" the message that they receive, and by doing that they bind additional variables. For example, if the OAA weather agent receives a predicate like the following.

weather(newyork,Forecast)

It interprets `newyork` as a constant and `Forecast` as a variable. The work of the agent is to find a value for Forecast that validates the predicate. In this process, the agent binds the variable `Forecast` to the weather forecast of New York City. The weather agent responds to the requesting agent with the same predicate with the bound variable `Forecast`.

RETSINA uses a functional representation, so the predicate is translated in a representation similar to

weather(newyork)

Here `weather` is used as a function applied to the constant `newyork`. The RETSINA weather agent computes this function and it returns the weather forecast that is then forwarded to the requesting agent.

The distinction between predicative use and functional use of the message is not only an abstract distinction. For example the following two predicates p(a,Y) and p(X,b) produce different information and they are mapped onto different functions. The first one takes `a` as input and produces a value for `Y`; while the second given an input `b` produces a value for `X`. These two predicates should be bound to two different functions: $p_1(a) \rightarrow Y$ and $p_2(b) \rightarrow X$. To translate OAA messages into RETSINA messages, the RETSINA-OAA Inter-Operator needs to know how the OAA predicate is used; specifically, it needs to know which arguments should be specified as inputs, and which arguments will be set as outputs.

Different Interpretation of Performatives As noted in [5] there is no one to one correspondence between the performatives used in KQML and the predicates used in OAA. For example, OAA has performatives like `ask-N` which is intended as asking for N possible solutions of the the query. On the other hand KQML has either `ask-one` or `ask-all`, ie it can ask only one solution or all solutions.

Despite the radical differences between the languages used by the two systems, a translation mechanism should be provided to allow agents in the two systems to communicate with each other. The following sections describe how problems were overcome.
The translation process is divided in three parts. First the RETSINA-OAA InterOperator needs to translate agent advertisements so that the services provided by agents entering the community can be shared across system boundaries. Second, the interoperating agent should translate queries across the systems. Third, the interoperator should translate the answers to the queries.

5.1 Translation of Advertisements

OAA advertisement and RETSINA advertisements express different information; OAA specifies the format of the queries answered by the agent, but not what information the agent needs to solve the query. Conversely, RETSINA specifies the information required by the agent, but not the format of the query (with the only exception of the information agents that have a standard query format.)
For example, the following predicate shows an advertisement of the OAA weather agent in OAA format.

weather(Place, Forecast)

An advertisement in OAA is just an example of query to which the agent replies. It does not specify what information the agent needs to compute the reply.

The advertisement of a RETSINA agent that reports the weather is displayed in figure 2 below.

```
(advertisement
  :name “handleSingleShotQuery”
  :ontology “weather”
  :inputVariables
    :name “primary-keys”
    :fields (listof (field “city” “string”))
    :attributes (listof )
  :outputVariables
    :name “output”
    :fields (listof
              (field “time” “string”)
              (field “weather” “string”)
              (field “weather-url” “string”)))
```

Figure 2: The advertisement for the RETSINA weather agent

RETSINA advertisements are complex, but for the scope of this paper the important fields are `ontology` that specifies the field of application of the agent, `outputVariables` that specifies what types of information the agent reports, and `inputVariables` that specifies what the agent needs as inputs in order to compute the output.
The translation of OAA advertisements into RETSINA advertisements requires the specification of which arguments of the OAA advertisement are used as inputs and which ones are used as outputs, and a mapping of arguments and positions into keys. The translation process maps the *functor* of the predicate in the *ontology*, the input variables into the *inputVariables* and the union of input and output variables into the *outputVariables*. The rationale behind this mapping is that the functor specifies the objective of the agent (what kind of information is produced by the agent), similarly the ontology field in the RETSINA advertisement specifies the domain of the agent. The input arguments of the OAA advertisement are mapped into the corresponding input variables of the RETSINA advertisement. Finally, a query to the agent results in the binding of all arguments of the predicate, therefore all arguments are used as output variables.
The mapping from RETSINA to OAA follows a similar schema: ontology is mapped to the predicate, inputVariables are mapped to input arguments and outputVariables are mapped to output arguments. Input variables that appear also as outputs are used only once.
Still, the mapping presented so far is not sufficient to translate OAA advertisements into RETSINA advertisements, since RETSINA advertisements require the specification of the ontological type and the data type of the information provided to the agent. For instance, the advertisement above specifies that the input to the RETSINA weather agent be of ontological type *city* and data type *string*. The role of this specification is twofold: on one side any agent that requests a service can make sure that the information provided to the advertised agent is consistent with what that agent expects; in addition both the ontological type and the data type are used by the Matchmaker to match requests and advertisements.

To match requests and advertisements, the RETSINA Matchmaker compares the advertisement and the request and tries to verify whether the input variables of the request match the input variables of the advertisement, and the output variables of the request match the output variables of the advertisement. The match of variables is done in two steps: first the Matchmaker does an exact match on the data types, second the Matchmaker uses an ontology based on WordNet [2] to match the ontological type of the request with the type of the advertisement. For instance, by using the ontology the Matchmaker recognizes that `(city string)` and `(place string)` match, but it would fail to recognize a match between `(city string)` and `(dog string)` because `city` and `dog` are ontologically distinct.
To summarize, the translation of advertisements between OAA and RETSINA minimally requires the following information:

- a description of which arguments in the OAA advertisement are used as input, and which ones are outputs;
- a definition of ontological type and data type of the information provided to the agent and information outputted by the agent. Finally a translation of the ontology in the RETSINA advertisement to match functors in OAA advertisements;
- a translation of terms from one ontology to the other, so that the Matchmaker recognizes that the OAA predicate "find" in the context of searching flight information, should be translated into the RETSINA "flight";
- a description of how the arguments of OAA predicates match the keys of RETSINA advertisements;

5.2 Translation of Queries and Answers

Advertisements are descriptions of the services provided by the agent. Advertisements are used by the middle agents to identify which agent provides a specified service. Once the provider is found, the requesting agent still needs to query the provider to obtain a service. While the protocol used by RETSINA agents differs from the protocol used by OAA agents, the essence of the process remains the same. One of the tasks of the RETSINA-OAA InterOperator is to translate queries sent by RETSINA agents in a format that can be understood by OAA agents.

```
(performative
 :ontology some-ontology
 :content (...))
```

Where the performative is either `tell` or `ask-one`. `Tell` is used for commands to agents to perform some action, while `ask-one` is used for requests to information agents.
The basic mapping follows the same rules used to translate the advertisements: the ontology is mapped in the functor of the corresponding OAA predicate and the content is used to map to the arguments of the predicate. Since the format of the content field depends on the type of agent queried, each query requires a specific translation rule.
For example, in the RETSINA system, information agents are constructed in an homogeneous way, furthermore they respond to queries with the same format. An example of these queries is the following:

```
(ask-one
 :ontology weather
 :content (objective :name "getInformation"
                     :parameters
                       (pval "primary-keys"
                             "(PittsburghPA)")))
```

The mapping for information agents requires the ontology to be mapped on the functor of the predicate while the parameters are mapped on the variables. In addition, output variables that are not specified in the query should be added to the predicate.
The `:parameters` field stores different information depending on the query. For the weather agent it contains the name of the city of the forecast; for the flight agent it contains a specifically formatted list with the values of a the start and end location of the flight and the leaving date and time.

6. AN EXAMPLE

The RETSINA-OAA InterOperator has been used as the interoperator agent between the RETSINA and OAA systems in the context of a MAS that assists humans to plan a NEO[2] operation. Figure 3 shows the agents used in this system. The RETSINA-OAA InterOperator is in the center of the picture, on the left side there are the RETSINA agents, while on the right are the OAA agents.
Upon joining the system, each agent advertises with their middle agent, either the OAA Facilitator or the RETSINA Matchmaker, depending on whether the agent belongs to OAA or RETSINA. The RETSINA-OAA InterOperator monitors the advertisements received by the middle agents, translates them and sends them as if it were own to the middle agent of the other system. Through this mechanism, the RETSINA Matchmaker receives the advertisements of all the OAA agents, while the OAA Facilitator receives all the advertisements of the RETSINA Matchmaker.
During the execution of the scenario, one of the human users requests weather information. The interface agent of the user sends a request to the Matchmaker that identifies two agents that report the weather: one is the RETSINA weather agent, that gathers weather information by reading the current page on the CNN web site, the other is the RETSINA-OAA InterOperator that advertised the capability of the OAA weather agent.
The interface agent of the user can query both agents to gather weather information. Either the RETSINA weather agent or the RETSINA-OAA InterOperator can be contacted because they advertised similar services. Whereas the RETSINA weather agent computes weather information directly, the RETSINA-OAA InterOperator translates the query to the OAA format and forwards the query to the OAA Facilitator, which in turn contacts the OAA weather agent to solve the query.
The results of the queries follows the opposite direction: the OAA weather agent sends its reply to the Facilitator, which

[2]A NEO (Non-combatant Evacuation Operation) is executed by the US State Department to evacuate civilians from areas in which an unrest threatens their lives

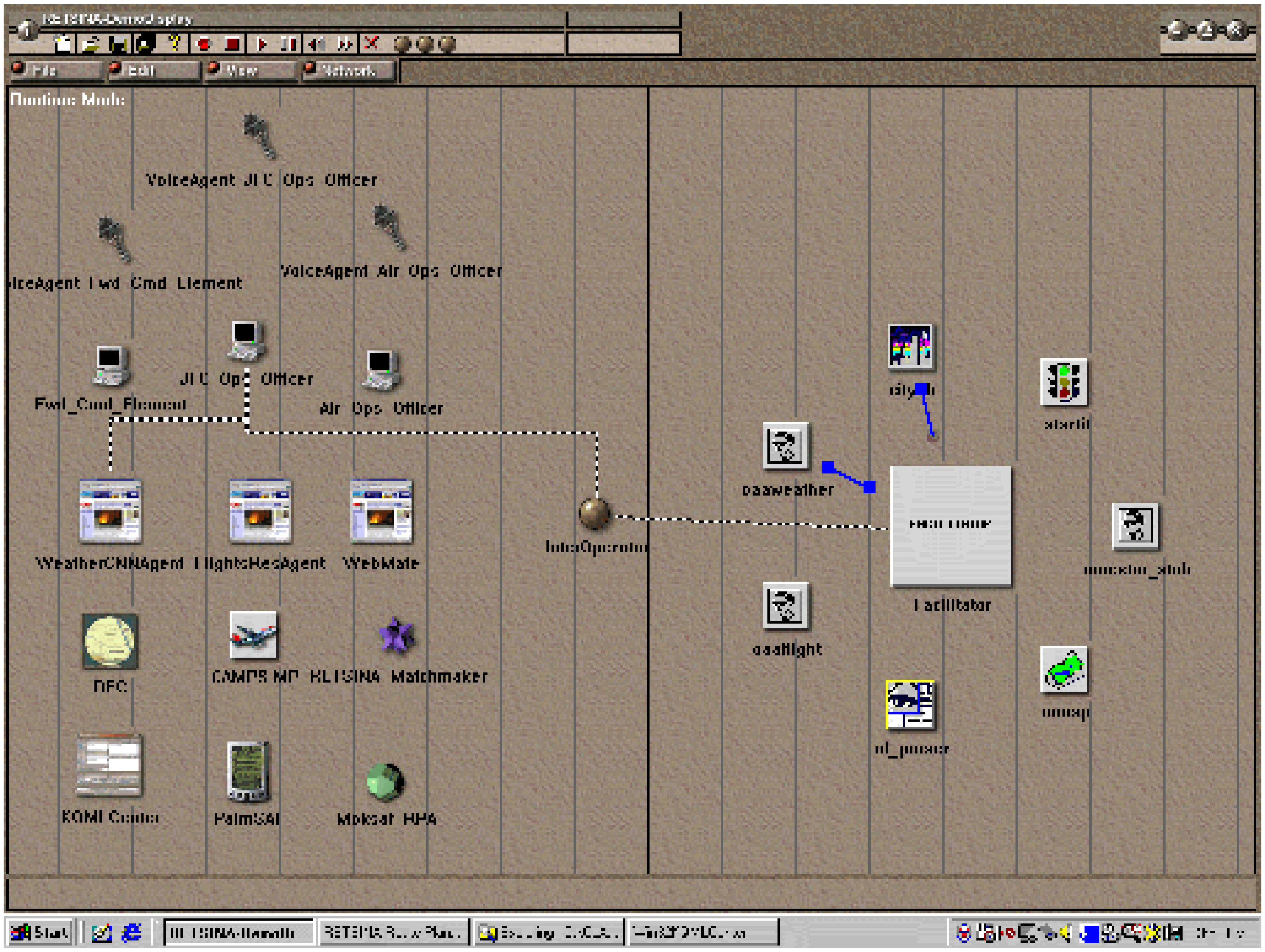

Figure 3: The RETSINA-OAA InterOperator forwards the "oaaweather" agent's *error* reply to the "JFC Ops Officer" agent, which then submits the same request to the "WeatherCNNAgent"

in turns sends the result to the RETSINA-OAA InterOperator. The data is then translated to the KQML format used by RETSINA agents and is sent to the interface agent for display to the user.
The two weather agents are interchangeable and they can be used as alternatives for the same query. If one of the two queries fails, for instance because one agent does not have weather information for the city requested, the query is automatically redirected to the other agent.

7. PRELIMINARY EVALUATION

To estimate the performance of the RETSINA-OAA InterOperator we ran the NEO scenario, we computed the number of messages exchanged and we logged the time that took the messages to be translated and transfered from one system to the other.
The agents used in the experiment are displayed in figure 3. There were 15 RETSINA agents and 9 OAA agents. In the scenario used for the experiment, the agents exchanged 162 messages, 50 of which where transmitted through the Interoperator. On average, message translation (i.e. the translation of queries, answers and RETSINA advertisements) took approximately 0.25 seconds, which is a negligible time considering that many agents performed slow operations like gathering information from web sites.
The only performance problem emerged with the translation of the OAA advertisements. The RETSINA-OAA InterOperator took 3 to 7 minutes to translate and advertise all the OAA agents due to the complexity of the data structures used by Prolog to represent those advertisements and due to the size of those advertisements (approximately 14 KB). This is only a one time problem, however, since it is unlikely that all the agents advertise (or unadvertise) at the same time during the execution of the scenario.

8. RELATED WORK

Interoperation across boundaries of multiagent systems is likely to become an important issue as the field of heterogeneous MASs matures and more MASs, based on different principles and ideas, are implemented. Indeed, another interoperator, called *OAA-KQML-Bridge* [4] was developed in parallel with ours. OAA-KQML-Bridge connects

the TEAMCORE agent [6], "TeamQuickSet", to the OGI [3] OAA agents. The OAA-KQML-Bridge implements a bidirectional KQML-OAA message translator that mediates between TEAMCORE and the OGI OAA Facilitator.
While the OAA-KQML-Bridge is similar to the RETSINA-OAA interoperator in the translation of OAA and KQML messages, the two implementations of the interoperator are nonetheless different in three respects. First, due to the lack of a TEAMCORE matchmaker, the OAA-KQML-Bridge did not attempt the problem of representing advertisements from one community to the other. Second, while in principle the OAA-KQML-Bridge's design is generalizable to handle the communications of agents in both communities, it was implemented to handle only specific dialogues with "TeamQuickSet". Finally, the OAA-KQML-Bridge maintained a connection state based on both the type of agent message (i.e. query vs. reply) and on the content of the agent message, to the extent that it would send a message to "TeamQuickSet" when all the information it required was ready. The RETSINA-OAA InterOperator, however, maintained connection state solely at the agent message level in the interest of "MAS Interoperational Transparency" (see principle #5, above).

9. CONCLUSIONS

In this paper, we presented an implemented agent that allows interoperability across MAS. The need for agents that facilitate the interoperation across MAS boundaries emerges with the increase of the number of agent communities on the Internet. The RETSINA-OAA InterOperator, presented in this paper, facilitates the communication between two implemented MAS: RETSINA and OAA. The RETSINA-OAA InterOperator allows agents in the two systems to communicate transparently, exchange services and be notified of the ever-changing availability of agents in each other's systems, with a minimum of overhead.

10. REFERENCES

[1] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of IJCAI97*, 1997.

[2] C. Fellbaum. *WordNet: An Electronic Lexical Database.* MIT Press, 1998.

[3] T. Finin, Y. Labrou, and J. Mayfield. Kqml as an agent communication language. In J. Bradshaw, editor, *Software Agents.* MIT Press, 1995.

[4] M. Huber. Personal Communications.

[5] D. Martin, A. Cheyer, and D. Moran. The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1-2):92–128, 1999.

[6] D. V. Pynadath, M. Tambe, N. Chauvat, and L. Cavedon. Toward team-oriented programming. In N. Jennings and Y. Lespérance, editors, *Intelligent Agents VI — Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Berlin, 2000.

[3] Oregon Graduate Institute

[7] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert, Intelligent Systems and their Applications*, 11(6):36–45, 1996.