Agent Communication Transfer Protocol

Alexander Artikis, Jeremy Pitt and Christos Stergiou
Intelligent & Interactive Systems Group
Department of Electrical & Electronic Engineering
Imperial College of Science, Technology and Medicine
Exhibition Road, London, SW7 2BT, UK
+44-(0)207-59.46316
{a.artikis, j.pitt, c.stergiou} @ic.ac.uk

ABSTRACT

The idea of agent-oriented middleware is to support high-level communication between computational processes like agents. Such communication requires an interaction protocol, an agent communication language and a transfer protocol. This paper focuses on the third requirement and presents the framework, design and implementation of an Agent Communication Transfer Protocol (ACTP). The ACTP is an application layer protocol designed to facilitate communication between heterogeneous agents. In particular, we show how the ACTP supports various ways of communication, logical abstraction of the communication process, platform-independence, a naming convention, reliability in message transfers and a degree of flexibility. In conclusion, we suggest that the ACTP contains the right reliability, flexibility and generality to support high-level interoperability in agent communications and therefore can be useful for standardization in FIPA compliant platforms.

Keywords

Agent-Oriented Middleware, Agent Communication Protocols, Multi-Agent Systems, Software Agents, Agent Conversations, Transfer Protocols.

1. INTRODUCTION

Communication is the *sine qua non* of multi-agent systems. Inter-agent communication requires knowledge of an interaction protocol, an agent communication language (ACL), and a transfer protocol [1]. In previous work, we developed a general framework for designing ACLs and interaction protocols [10], [11]. In this paper, we concentrate on the third requirement, the transfer protocol.

Our starting point is the observation that humans use multiple channels for communication, and if one fails, we try another (e.g. with no reply to a phone call, we may send e-mail). If we seek to support higher-level communication between agents rather than processes, some of the same flexibility should be seen, without forgetting that we are essentially dealing with a mechanistic interaction.

Reviewing different alternatives for combining high-level flexibility with mechanical efficiency, we find that:

- network protocols or protocols in distributed computing are too inefficient for the purposes of intelligent agents [1];
- although a number of different agent frameworks are available, specifically designed for inter-agent communication, there appears to be no architecture that combines alternative ways for message and data transfer in an efficient way;
- almost all protocols impose a number of constraints on the agents that can use them, requiring agent-level customization on the part of the developer.

Our proposed solution is the Agent Communication Transfer Protocol (ACTP), which is an application layer protocol that combines various features of existing communication protocols with a number of innovative ones. The purpose of implementing the ACTP was to support agent-oriented middleware with the following facilities:

- *Support for various kinds of communication and data transfer*. Agents are able to interact in a synchronous as well as an asynchronous manner. The choice of having conversations in a one-way or a bi-directional fashion is offered.
- *Logical abstraction of the communication process*. Agents view the whole data and message transfer process as a black box. In addition, agents that use the ACTP run on any platform and no constraints are imposed on their functionality.
- *The use of a name*. The agent that sends a message denotes the peer agent not by its physical address, but by its name. It is up to the transmission system to convert this name to a physical address.
- *Reliability of transfer*. The ACTP is resilient to failures and it informs the agents with appropriate messages about each type of error that may occur. In the case of multiple failures, the ACTP uses many different methods of communication to ensure a successful data transfer.
- *A degree of flexibility in the handling of interactions*. The ACTP will take into account the history of previous interactions when dealing with future communications.

In section 2, we define a general framework of the Agent Communication Transfer Protocol. Section 3 describes the design of the ACTP using UML. Based on this object-oriented design,

section 4 illustrates the implementation of this protocol along with its special features, like the abstraction of the communications, the ability of direct interactions and the use of alternative protocols in the case of failures. Section 5 contains the evaluation and testing of the ACTP illustrating its visualization and a failure scenario. Finally, we discuss related and future work and draw some conclusions. These include the standards that the ACTP has set and indicate that agent-oriented middleware should support a significant degree of flexibility, reliability and interoperability.

2. THE GENERAL FRAMEWORK

The details of the networking layers that lie below the primitive communicative acts of the ACLs are hidden from the agent. The ACTP manipulates the application protocols taking under consideration the dependencies that exist among them. The protocols that the ACTP uses in order to carry out the agent communication have several relationships between them. These relationships play a key role in the design of ACTP and are illustrated in figure 1.

The bottom layer represents all protocols that the hardware provides. The second layer from the bottom is the Internet Layer and contains the IP or IPv6. It also includes the error and control message protocol ICMP. The two basic protocols of the Transport Layer are the TCP and the UDP. The *Real Time Protocol* (RTP) and the *Resource Reservation Protocol* (RSVP) also reside in the same layer. As presented in figure 1, the Application Layer illustrates the dependencies among the various application protocols. Each enclosed rectangle corresponds to an application protocol and resides directly above the protocols that it uses.

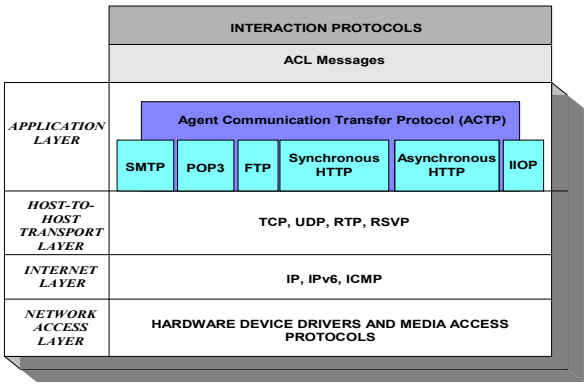

Figure 1: Location of the ACTP in the TCP/IP model

The application protocols that the ACTP uses as transport mechanisms use other protocols in order to carry out their tasks. In other words, many of the application protocols can be viewed as a wrapper, in the sense that they hide the implementation and use of other protocols from the ACTP. For example, the FTP uses the synch signal of Telnet to abort an in-progress transfer. Nevertheless, the ACTP does not interact with Telnet; it just uses the FTP and it does not deal with the way the FTP is implemented. In the architecture of the ACTP there are two levels of encapsulation; in the higher one, the networking details of the communication processes are hidden from the agents. In the lower one, the implementation of the underlying application protocols is concealed from the ACTP.

In this layered architecture, the ACTP resides in the Application Layer and, in particular, it is above all the other application protocols. In other words, the ACTP provides an interface between the ACL messages and the Application Layer. In this way, the ACTP receives the communicative acts through the ACL messages and uses the appropriate underlying protocol for the actual data transfer. The ACTP could not be a transport protocol because the primary duty of these protocols is to transfer data from one application program to the other. The ACTP's primary role is to allocate an application protocol to carry out the agent communication. If the communication is not successful, then the ACTP will decide which protocol will be the next one to use. If the ACTP were a transport protocol, it would not be able to allocate an application protocol as the means of communication between agents.

The Agent Communication Language (ACL) messages reside above the Application Layer. Finally, above the ACL messages are the interaction protocols. As shown in figure 1, the ACTP provides the API between the network protocols and the agent execution framework, hiding all the low-level communication details from the agent. The ACTP integrates the agent's speech acts with the network mechanisms that implement them, providing an advanced form of communication in multi-agent systems.

3. THE ARCHITECTURE OF THE ACTP

3.1 The Conceptual Model of the ACTP

The conceptual model is an abstract representation of the concepts in the problem domain. In this case, the basic concepts (components) of the Agent Communication Transfer Protocol are the NameServer, the Agent Interface, and the underlying communication mechanisms. The structure of these components is illustrated in the conceptual model in figure 2. It is important to note that the term conceptual model strongly emphasizes domain concepts and not software entities or classes (the software classes are described in section 3.2.2).

3.1.1 The NameServer

The NameServer is a process that provides the ACTP with low-level information about the agents, such as their logical name and domain, and their physical addresses in relation to each underlying protocol (e.g. the physical address of an agent concerning FTP may be ftp.ee.ic.ac.uk). In addition, the NameServer stores authentication data that may be needed in order to contact each agent (e.g. an agent's FTP account is accessible only after providing a valid user name and password). The NameServer acquires this information when the agents *register* themselves in the ACTP and stores it in its database. In addition, the NameServer stores the names of the underlying protocols that the ACTP uses. In this way, the queries that agents make to the ACTP about the protocols that can be used for communication are forwarded to the NameServer and the results are passed back to the agents.

Before establishing a connection between two agents, the ACTP "asks" the NameServer for the physical addresses of the two agents, as well as any authentication information that has to be used in order to contact them. If an agent provides this low-level information then the NameServer will not be contacted. The function of the NameServer of the ACTP resembles the one of the facilitator of KQML and the Agent Management System of FIPA. The basic difference between the NameServer and KQML's facilitator is that the latter entity also functions as a matchmaker, providing information about the abilities of agents. The NameServer does not store such information because the ACTP is

a communication protocol and not an interaction one. In other words, the ACTP provides an efficient data transfer mechanism and does not deal with the social context of the communicating agents. The relationship between the FIPA platform and the ACTP will be thoroughly discussed in section 6.

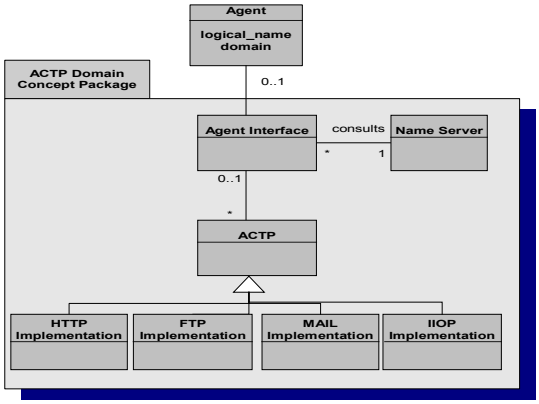

Figure 2: The Conceptual Model of the ACTP

3.1.2 The Agent Interface

The main component of the Agent Communication Transfer Protocol is called the Agent Interface. It receives requests from the agents and, after processing them, it forwards them to the appropriate ACTP modules. Then, it informs the agents of the outcome of their request. The Agent Interface is the main component of this protocol that each agent interacts with. The agents are not aware of the existence of the NameServer and are not concerned with the actual implementation of the underlying protocols. The Agent Interface module is also responsible for the error detection and handling of the Agent Communication Transfer Protocol (see section 4.3).

3.1.3 The Communication Mechanisms

The ACTP is organized into a generalization-specialization type hierarchy (or type hierarchy). The super type ACTP represents a more general concept (i.e. a general way of communication) and the subtypes HTTP Implementation, FTP Implementation, MAIL Implementation and IIOP Implementation represent more specialized ones. In this case, identifying super and subtypes is of significant value because their presence allows the understanding of concepts in more general, refined and abstract terms [7].

3.2 The Design of the ACTP

3.2.1 The Semantics of the Agent Communication Transfer Protocol

The basic commands that an agent will use concerning the ACTP are *read*, where the agent requests to receive data, and *write*, where the agent requests to transmit data. Before *reading* or *writing* the agent will either declare to the ACTP which protocol it wants to be used for the data transfer or it will let the ACTP decide which protocol is best suited for that transfer. The remaining commands of the ACTP provide the agents with the ability to be informed about the available underlying protocols, to register and deregister themselves to the ACTP's database, and to change their IP address when moving from one host to another.

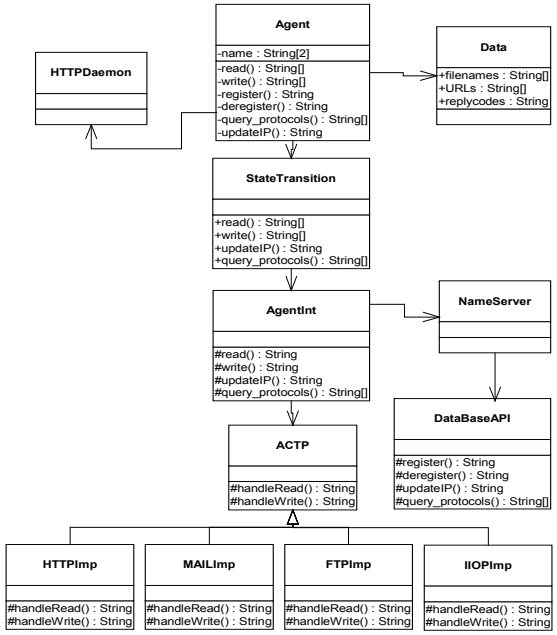

Figure 3: The Design of the ACTP

3.2.2 The Class Diagram of the Agent Communication Transfer Protocol

The class diagram in figure 3 represents the basic architecture of the Agent Communication Transfer Protocol. In this phase, we have transformed the concepts of the ACTP domain into software classes. For example, the *StateTransition*, *Data*, and *AgentInt* classes represent the concept of the Agent Interface, while the *NameServer* and *DataBaseAPI* classes represent the concept of the NameServer.

The *Agent* class represents an agent that uses the ACTP and contains a number of methods that must exist in all agents that want to communicate through the ACTP. The agent provides the *StateTransition* class with data in the form of the attributes that are defined in the *Data* class. The *StateTransition* class implements the error handling mechanism of the protocol. This class forwards the agent request to the *AgentInt* class, which, in turn, queries the NameServer about low-level information that concerns the communicating agents. The *AgentInt* class handles the agent requests and formats the messages that are sent to the NameServer through the TCP. In addition, the *AgentInt* class passes the agent requests along with the low-level information to the *ACTP* class. The *ACTP* class provides two abstract methods, called *handleRead* and *handleWrite,* which are inherited by its subclasses. The subclasses of the *ACTP* class are the *HTTPImp* (Imp is short for implementation), the *FTPImp*, the *MAILImp* and the *IIOPImp* classes, and they forward the agent request to the actual application protocols. The class diagram in figure 3 is simplified and does not illustrate the whole design structure of the ACTP. Each subclass of the *ACTP* class implements the actual data transfer in the context of an application protocol or a communication mechanism. The design of each communication technique is not represented in figure 3 in order to keep the class diagram comprehensible.

The agent also interacts with the HTTP Daemon that the ACTP provides it with. We will describe in detail the concept of the HTTP Daemons in the next section. The *HTTPDaemon* class represents the HTTP Daemon and it communicates with the agent through TCP messages.

The structure of the ACTP resembles the structure of the *state* design pattern [4]. The *read* and *write* methods that the agent invokes result in the invocation of the ACTP's *handleRead* and *handleWrite* methods. These methods are inherited and overridden by the *ACTP*'s subclasses that carry out each specialized way of communication. The *handleRead* and *handleWrite* method invocation is manipulated by polymorphism of the ACTP subclasses and not by the programmer. The benefits of having such a structure are several:

- Each protocol implementation is put into a separate class. New protocols for the agent communication can be added by defining new *ACTP* subclasses. The state pattern localizes application-specific behavior and partitions behavior for different protocols.
- Size and complexity of operations is reduced, enhancing code understanding and maintainability.
- Adding a new protocol is wholly independent of the agent. An agent simply invokes a *read* or *write* command. The way this method will be handled is independent of the agent. In this way, adding or removing a protocol from the ACTP will not result in modifying the agent code.

4. IMPLEMENTATION

This section describes the implementation of the Agent Communication Transfer Protocol design that was illustrated in the previous section. Java was chosen as the main implementation language of this protocol because it supports platform-independence, network programming, parallelism, high-level client-server programming, connectivity to databases, as well as graphical user interface production. In this section we illustrate how the implementation supports the facilities described in section 1.

4.1 Support for Various Kinds of Communication and Data Transfer

4.1.1 Indirect and Direct Interactions

The Agent Communication Transfer Protocol supports both indirect and direct interactions. In the first case, the communicating agents are not aware of the physical addresses of their peers. The ACTP module of the initiating agent will consult the NameServer about that necessary information and it will use it in order to contact the peer agents. Such a scenario is demonstrated on the left-hand side of figure 4. The numbers in this figure show the sequence of the exchanging messages.

If the agents know the physical addresses and authentication information of their peers in advance (i.e. user names and passwords), then the ACTP module will not contact the NameServer, and the interaction will be direct between the ACTP modules of the communicating agents. This way of communication is demonstrated on the right-hand side of figure 4. The communication between two or more ACTP modules and between an ACTP module and the NameServer is performed through the TCP. On the other hand, the interaction between an agent and its ACTP module is performed through method invocation.

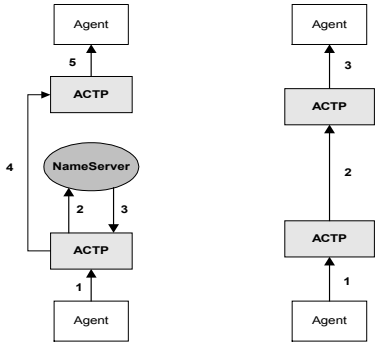

Figure 4: Indirect and Direct Communication Using the ACTP

4.1.2 Use of Various Ways of Communication

The ACTP uses a number of different transport mechanisms in order to accomplish the data transfers. The ACTP uses two forms of the HTTP; the first one is called *SynchronousHTTP* and the second one is called *AsynchronousHTTP*. The SynchronousHTTP functions almost in the same way as the HTTP. The client contacts the web server of the peer and receives the web pages it requests. The reason that this way of communication is called SynchronousHTTP and not just HTTP is to contrast it with a similar but asynchronous way of communication that we call AsynchronousHTTP.

The Agent Communication Transfer Protocol provides each agent with an HTTP Daemon, which is a server that receives all the AsynchronousHTTP messages of the agent it belongs to and stores them for a certain period of time. If the agent does not check for its messages during that period of time or checks but does not accept them, then its HTTP Daemon will delete them. When using the AsynchronousHTTP in the context of the ACTP, an agent sends a message to a peer and does not wait for a certain period of time to receive a response, as it happens with the SynchronousHTTP. As already explained, the HTTP Daemon of the peer agent will receive and store this message. When the peer agent is informed that it has a new message then it may either accept it or not. If it does accept it then it may either respond or not. In the case that the peer agent responds, it may send the response at any time. This kind of communication is asynchronous, and the format of the exchanging messages is similar to those of the HTTP. This is why the communication protocol is called AsynchronousHTTP. It is important to note that this way of communication can become synchronous if the peer agent responds in real-time.

The remaining application protocols that the ACTP uses are the FTP, the SMTP and the POP3. The FTP is used for the transfer of large messages, while the SMTP is used to send e-mail and the POP3 to receive e-mail. We are currently working on an implementation of the IIOP in order to include the object-oriented technology in the ACTP (this work is described in section 6.2.1).

4.2 Logical Abstraction of the Communication Process

The agents that use the ACTP do not deal with low-level networking details at all. They only use a limited number of high-level commands in order to communicate. These commands offer a significant degree of encapsulation, hiding the implementation aspects of the communication process. In addition, using the

NameServer, the ACTP does not require the agents to know low-level information about their peer agents. Each agent can refer to the others using only their names. In other words, using the ACTP, inter-agent communication is performed in a high-level. Consequently, agents using the ACTP run in any platform and operating system. In general, very few constraints are imposed on the agents that use the ACTP; agent interactions can be performed only with the use of a *read* and a *write* command.

4.3 Reliability of Transfer

Each underlying protocol and communication technique has its own error handling mechanism. The ACTP's error handling mechanism is based on the outcome (failure or success) of the execution of the application protocols. This error handling mechanism is based on a state transition diagram where the different states represent the various protocols that can be used to implement the agent's interactions, and the events represent the outcome of these interactions. Figure 5 demonstrates a partial state transition diagram that represents the way the error handling mechanism of the ACTP works. A partial diagram is shown in order to be comprehensible.

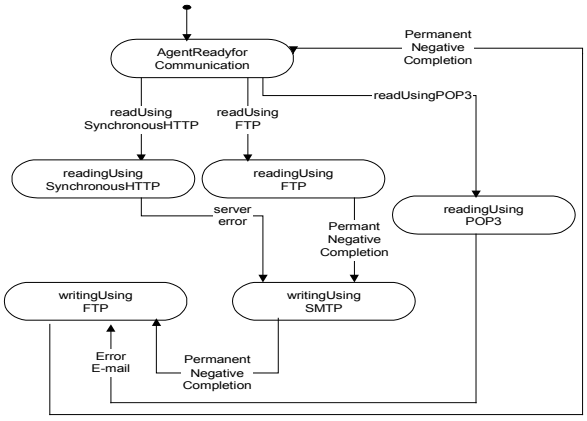

Figure 5: The State Transition in the Case of Failures.

The occurrence of an event results in a transition from one state to another. For example, when the ACTP is *writing* using the SMTP and a permanent negative completion occurs, then the ACTP will send a file using the FTP to the peer agent, informing it of the attempt that was made to contact it. In addition, the agent that invoked the *write* command will be notified (with the appropriate reply code) that the attempt to use the SMTP failed and that the ACTP has tried to use the FTP for the data transfer.

In the state transition diagram we represent only the state transitions that occur in the presence of permanent failures, meaning that there is no way to contact the peer using a particular protocol. The temporary failures are not represented because they are handled by each protocol until they become permanent. Finally, failures that are caused by agents (i.e. agents providing the ACTP with incorrect or missing data) are simply reported back to them and are not dealt with.

The process of using a new protocol for the data transfer (in the case of failures) continues until all the appropriate protocols have been used (according to the state transition mechanism), or the data transfer has been carried out successfully. Only then will the agent be informed about all the communication attempts.

Figure 6 illustrates the case where the attempt of a data transfer has resulted in a permanent negative completion. In this scenario, when the StateTransition instance receives the reply code indicating the failure, it uses the next appropriate protocol in order to carry out the data transfer that the agent requested. The sequence of messages is the same in the new attempt but now the AgentInt instance asks the NameServer about physical addresses of agents concerning another protocol. If the first attempt to transfer data had not resulted in a failure then the StateTransition would not have triggered an execution of a new protocol.

This state transition mechanism resides on the agent-side and can be configured by each agent. In other words, each agent can customize the state transition according to its preferences and modify it at any time.

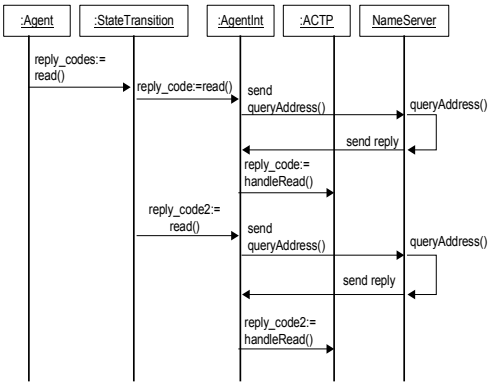

Figure 6: The Sequence Diagram of a *read* Invocation in the Case of Failures.

4.4 A Degree of Flexibility in the Handling of Interactions

Normally, an agent will specify the exact protocol that it wants to be used for the data transfer. In this case the ACTP will use the specified protocol for the communication. In other cases, an agent might not specify a particular transport mechanism and let the ACTP decide about that. The ACTP has the flexibility to take that kind of decisions. In particular, the ACTP offers a decision-making algorithm that is primarily based on the properties of the message being exchanged, like the size and the time-criticality of the message. The output of this decision-making algorithm is the transport medium that will be used for the message transfer. We discuss ways of enhancing the decision-making algorithm in the future work section.

5. TESTING AND EVALUATION

The ACTP was developed on the Windows NT platform and has been tested on Windows 98 and Linux platforms.

The purpose of producing a GUI for the ACTP was to explain the use of this protocol to developers who are going to use it as the communication medium of the agents they create and control. The basic window of this GUI is shown in figure 7. This GUI illustrates the facilities of the ACTP (e.g. the transport mechanisms, registration of agents etc.) and the way these facilities can be used. In real applications agents will interact with the core ACTP code like the GUI code interacts with it.

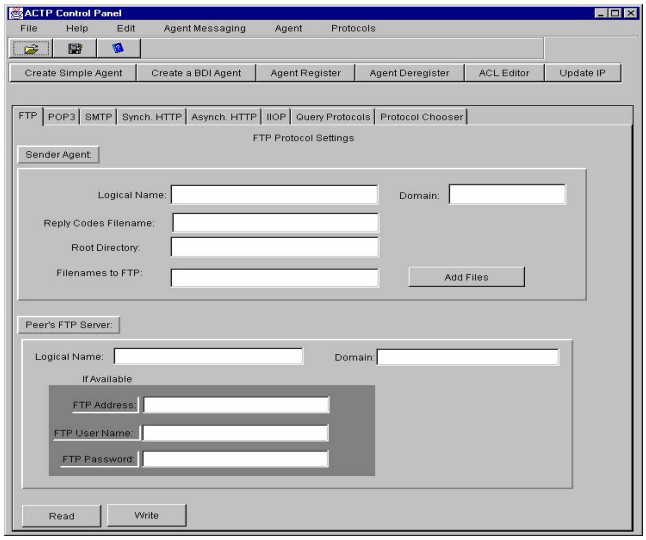

Figure 7: GUI of the ACTP

We will illustrate a scenario in order to show the way the ACTP handles failures. In this scenario (figure 8), an agent requests its ACTP module to retrieve some web pages from the peer's web server (invocation 1). The ACTP module consults the NameServer about the http address of the peer (message 2) and uses the SynchronousHTTP in order to retrieve the requested web pages (message 3). Supposing that a permanent failure occurs during the execution of the SynchronousHTTP, the ACTP module will use the next appropriate protocol (based on the state transition algorithm) which is in this case the SMTP. In other words, the ACTP will send the peer an e-mail requesting the web pages that could not be retrieved (message 5). Similarly, if an additional failure occurs in the execution of the SMTP, the ACTP will use the FTP in order to send the peer a file containing the requested web pages (message 7). Before each protocol execution, the ACTP will consult the NameServer about the physical address of the peer concerning that protocol (messages 2, 4, 6). In figure 8 the dashed lines represent the fact that at any time the peer agent can invoke a *read* command concerning its servers or ACTP module.

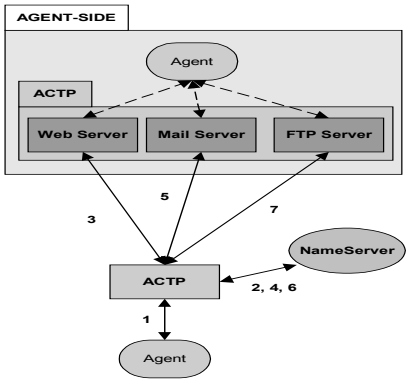

Figure 8: The ACTP in the Case of Failures

The model that is most commonly used to evaluate middleware systems measures such products against the following dimensions [5], [9]:

- *Availability*. The ACTP is robust concerning software failures. All possible *exceptions* are *caught* and appropriately dealt with.
- *Reliability*. The ACTP provides a good degree of reliability using the reply codes and state transition mechanisms. However, while the ACTP's fault management deals with network operations, higher-level failures (i.e. syntax errors in the ACL message) should be handled by the agents.
- *High Performance*. The use of multi-threading techniques makes the throughput of data transfers acceptable for a reasonable number of agent conversations.
- *Security*. The ACTP lacks security measures. We propose ways to overcome this deficiency in the future work section.
- *Scalability*. New agents can use this protocol without affecting existing agent interactions. New agents can use the existing ACTP modules (i.e. agents that reside on the same machine) or can download and use new modules.
- *Resource Handling*. The ACTP makes good use of the network resources, as it opens a new connection only when an existing one fails to achieve a message transfer.
- *Applicability*. The ACTP is platform-independent and provides simple *read* and *write* API functions.

Since it is not possible to evaluate middleware systems against objective criteria, we performed subjective assessments against the above dimensions. Our overall assessment of the ACTP is that it performs satisfactorily in these dimensions.

6. RELATED AND FUTURE WORK

6.1 Related Work

There are a number of platforms that provide a communication transport for multi-agent systems. Our work has similarities with some platforms and offers a number of innovative features that enhance the agent communications.

The Simple Agent Communication Protocol (SACP) [12] is an application protocol that aims at making agent conversations simpler for developers and works by grouping agent communication into two roles, listening and talking. Unlike the ACTP, the SACP does not provide peer-to-peer communication. In general, apart from the fact that the SACP is easy-to-use like the ACTP, the only significant ability that it provides is the well-structured data storage mechanism. Agents that use the SACP can exchange data that are in the form of this mechanism, which is called "nodespace".

The Java Agent Template, Lite (JATLite) [6] is an agent architecture that does not merely provide a communication protocol, but an agent model that uses a specified transport medium. The functionality of the communication protocol of the JATLite bears a strong resemblance to the functionality of the ACTP. Both the JATLite and the ACTP support asynchronous communication in a bi-directional as well as in a one-way manner and they both use a number of network application protocols like the FTP and the SMTP. Nevertheless, the JATLite model has a basic disadvantage. Each agent conversation has to go through the Agent Message Router (AMR). If the recipient of a message has crashed or is unavailable for some reason, then the AMR will store the message and will send it again when the recipient is available again. Clearly, under this scheme the AMR creates a

bottleneck. Due to the fact that the ACTP supports indirect as well as direct communications, the bottleneck that is created in the JATLite architecture has been overcome. In the case where the NameServer crashes, agent communications can still be performed, provided that the communicating agents are aware of the physical addresses of their peers.

Another disadvantage of the JATLite architecture is that, although it is constructed in modular layers, it is inefficient to include an IIOP implementation in a future version. The inclusion of an IIOP implementation would require the rewriting of the AMR, in order for it to be able to send and receive IIOP messages. On the other hand, due to the extensible design of the ACTP (using the state design pattern), any implementation of a new protocol can be included without having to perform major changes.

Aglets are Java objects that have the ability to move from one host on the Internet to the other and use the Agent Transfer Protocol (ATP) as the default implementation of the communication layer. The ATP is a simple, application-level protocol, designed to transmit an agent in a system-independent manner [8]. The ATP is modeled on the HTTP, providing an asynchronous message-passing mechanism using a client/server paradigm. The ATP also supports synchronous peer-to-peer communication between agents. The ATP is a rather restricted communication protocol because it is used mainly to enable mobile agents, and in particular the Aglets, to move from one host to the other. Due to the fact that the ATP does not provide a wide range of communication capabilities, the Aglets model also uses the RMI mechanism and CORBA in order to communicate. Mobility is supported in the ACTP through the FTP and the *updateIP* command. In particular, the code of an agent can be sent to a new host through the FTP and the moving agent can inform the ACTP using the *updateIP* command. The result will be to update its IP address in the NameServer's database and, consequently, all its messages will be sent to its new IP address.

The FIPA Agent Message Transport (AMT) [2] appears to be the most complete communication platform for multi-agent systems. The AMT has many similarities with the ACTP, like the support for both synchronous and asynchronous communications, the support for both direct and indirect interactions, and the logical abstraction of the communication process. Furthermore, the FIPA platform offers a number of facilities that the ACTP lacks, like the application of security mechanisms on agent communications (i.e. transport-level security and agent audit logs) and the support for brokering. Nevertheless, the ACTP is complementary to FIPA in a way, containing features that are not included in the FIPA platform. In particular, the ACTP is very reliable in the data and message transfers, due to the fact that it uses a number of alternative protocols in the case of failures. In addition, the ACTP provides a greater degree of interoperability, enabling communication between non FIPA-compliant agents, and thus providing an alternative route to legacy software integration.

6.2 Future Work

The present work is the first step in the development of an intelligent middleware that integrates heterogeneous agents. Further work is necessary in order to establish the ACTP as a standard middleware in multi-agent systems.

6.2.1 Integration of OO Technology With Network Protocols

The ACTP has been designed in such a way that any protocol can be added without having to perform any major changes to the existing code. We are currently working on an implementation of the IIOP. The IIOP provides location and programming language transparency, and will thus make the ACTP a very powerful tool in the area of agent communication. An integration of object-oriented technology with network protocols will offer features such as the concept of inheritance that object-oriented techniques contain, as well as the reliability of transfer and maintainable software that message-based middleware provide.

6.2.2 Use of a Learning Algorithm to Decide the Way of Communication

The decision-making algorithm of the ACTP chooses the transport mechanism based on criteria, which are static and do not change dynamically. In order to enhance this algorithm, we plan to modify it, so that it will also take the history of previous interactions into consideration. In particular, we plan to use a learning algorithm that will also take into account the outcome of previous data transfers when specifying the transport mechanism for future ones. In this way, the ACTP will learn from previous failures and will avoid repeating them in the future.

6.2.3 Application of Security Measures on the Communications

The data transfers through the ACTP are liable to a number of security threats. A way of grouping these threats can be in terms of active and passive attacks [14]. A form of an active attack is the altering of the messages in transit between two peer agents. Passive attacks include eavesdropping on network traffic. There are a number of security measures that could be applied in the future versions of the ACTP, in order to improve the level of authentication, integrity and confidentiality of the agent conversations. One of these measures could be the implementation of a security application just above the TCP. An example of this approach is the Secure Sockets Layer (SSL) [14]. An alternative would be to enforce security constraints on the IP level, by using the IPv6 and the IPsec. Short for IP security, IPsec was developed to support secure exchange of packets at the IP layer. The advantage of applying security constraints on the IP level is that it is transparent to agents and applications and provides a general-purpose solution.

6.2.4 Support for Programming Language Independence

Due to the fact that the ACTP has been implemented in Java, only Java agents can use this protocol. Using an Interface Definition Language (IDL) can lift such a constraint in the future. Using an IDL, agents can invoke the ACTP methods without knowing where the objects of the ACTP that they access reside, or in what language the requested objects are implemented. In other words, any agents written in a language that supports method invocation of an IDL will be able to use the ACTP.

6.2.5 Resource Handling

General middleware technologies do not offer load balancing or offer it only on a limited basis [13]. If an agent is unable to satisfy all requests, then the ACTP should have the intelligence to send these requests to an equivalent agent, if there is one. Such a

facility can be implemented by providing a fundamental brokering mechanism at the NameServer. In particular, the NameServer could store the type of requests each agent can process. In this way, the ACTP will consult the NameServer about alternative agents, when one is unable to satisfy a request.

The ACTP should also provide adaptation to traffic. In other words, it should be able to handle an increase in network traffic effectively (e.g. due to an increase of agents). Each ACTP module can check how many agents are connected to it and prioritize the way these agents will communicate.

6.2.6 Integration with FIPA-OS

As already mentioned, our work has many similarities and can be viewed in some ways as complementary to the FIPA communication platform. Our aim is to integrate the ACTP with an implementation of the FIPA platform and for that reason we plan to integrate our protocol with FIPA-OS [3] that has been produced by Nortel Networks. FIPA-OS was chosen for that integration because the source code is free and because it follows FIPA's specifications in detail.

The integration of the ACTP with that platform can be performed in various levels. In a lower level, we can integrate the application protocols that the ACTP uses with the IIOP that FIPA uses as the Message Transport Protocol (MTP). In a higher level, we can include the state transition and the transport decision-making algorithm in FIPA's Agent Communication Channel (ACC).

The Agent Management System (AMS) of the FIPA platform is an entity that, among other things, stores the mapping between globally unique agent names and local transport addresses, like the NameServer does. The NameServer also stores authentication information about agents concerning each transport mechanism, an aspect that can be included in the AMS.

7. SUMMARY AND CONCLUSIONS

In this paper we have presented an agent-oriented middleware that offers a number of novel features that enhance communication in multi-agent systems. The development of the ACTP was accomplished on the basis of an extensible and well-structured design. The ACTP provides various levels of encapsulation and handles conversations in an efficient manner, thus shielding the agents from the complexity of the communication process.

The study of related work in the agent communication area highlighted various innovative capabilities of the ACTP, as well as a number of features that this protocol lacks. The ACTP has been tested over a set of classical examples, like the negotiation process that is defined in the Contract Net Protocol. In particular, the test cases involved multiple failures of several components of the communication process. Due to the fact that a significant effort was spent on the implementation of the error handling mechanism, the ACTP has proved to be very reliable and robust.

Having developed the first version of the Agent Communication Transfer Protocol we suggest that it contains a significant degree of generality, reliability and flexibility to support agent conversations in heterogeneous environments. Therefore, the ACTP can be standardised as a communication platform for multi-agent systems.

In the short term, we plan to make the object code available on the web (http://www.iis.ee.ic.ac.uk/actp) and install the NameServer on a dedicated computer at Imperial College. We will also be able to demonstrate the full version of the Agent Communication Transfer Protocol at future events.

Finally, we plan to integrate the ACTP with FIPA-OS in order to produce a reliable, flexible and complete communication transport that will offer high-level interoperability in agent interactions.

8. ACKNOWLEDGMENTS

This work has been supported by UK EPSRC, Nortel Networks joint-funded project CASBAh (GR/L34440) and EU Esprit project MAPPA (EP 28831).

9. REFERENCES

[1] Finin T., Labrou Y. and Mayfield J. KQML as an Agent Communication Language. Baltimore, U.S.A., 1995, 1-22.

[2] FIPA. FIPA'99 Specifications: Agent Message Transport. 1999. http://www.fipa.org/spec/fipa9716.doc.

[3] FIPA-OS. http://www.nortelnetworks.com/fipa-os.

[4] Gamma E., Helm R., Johnson R. and Vlissides J. Design Patterns, Elements of Reusable Object-Oriented Software. Addison-Wesley Publishing Company, 1995.

[5] Gorton I., Liu A., Greenfield P., Tran P. Evaluating Object Transactional Monitors within MEP. Advanced Distributed Software Architectures and Technologies (ADSaT), CSIRO Mathematical and Information Sciences. http://standishgroup.com/mep.htm

[6] Heecheol J. JATLite FAQ. Stanford University. http://cdr.stanford.edu/ProcessLink/papers/JATL.html.

[7] Larman C. Applying UML and Patterns. Prentice-Hall, 1997, 1-15.

[8] Oshima M., Karjoth G. and Ono K. Aglets Specification 1.1 Draft. http://www.trl.ibm.co.jp/aglets/spec11.html.

[9] Ovum Evaluates: Middleware. http://www.ovum.co.uk/ovum/reports/enterprise_middleware.htm

[10] Pitt J.V. and Mamdani A. A Protocol-Based Semantics for an Agent Communication Language. *Proceedings of IJCAI'99*, Stockholm, Morgan-Kaufmann Publishers, pp. 486-491, 1999.

[11] Pitt J.V. and Mamdani A. Designing Agent-Communication Languages for Multi-Agent Systems. In F. Garijo and M. Boman (eds.): *Multi-Agent System Engineering: Proceedings MAAMAW'99*, LNAI1647, Springer-Verlag, pp. 102-114, 1999.

[12] Reilly D. Simple Agent Communication Protocol. 1999. http://www.davidreilly.com/sacp/.

[13] Serain D. Middleware. Springer-Verlag, London, 1999, 1-30.

[14] Stallings W. Cryptography and Network Security. Prentice-Hall, 2nd eds, 1999.