Communicating Agents in Open Multi Agent Systems

Terry R. Payne, Massimo Paolucci, Rahul Singh, and Katia Sycara

Carnegie Mellon University,
The Robotics Institute,
5000 Forbes Avenue,
Pittsburgh PA 15213, USA,
{terryp, paolucci, kingtiny, katia}@cs.cmu.edu,
http://www.cs.cmu.edu/~terryp/index.html

Abstract. Agents often utilise the services of other agents to perform tasks within multi agent systems. To achieve this, an agent must first locate another agent that has the capability to provide a desired service (i.e. a service provider agent), and then interact with it. To communicate with a service provider, an agent requires information about: 1) the service provider agent's interface; 2) the ontology that defines concepts used by the provider agent; and 3) the agent communication language (ACL) the agent uses so that it can parse and understand the communication. Currently deployed MASs encode the interface description and the ontology within the capability description of a service provider, but assume a common ACL between communicating agents.
Middle agents support the discovery of service providers based on the advertised capabilities of the service providers. This advertisement defines the provider agent's interface, and may reference the ontology used by the provider agent. However, the requester agent still requires information about the ACL used by the provider to be able to communicate with it. This paper demonstrates how agents can communicate with each other without making assumptions about the ACLs used, by presenting a template based shallow parsing approach to message construction/decomposition, thus greatly simplifying and improving the robustness of inter-agent communication.

1 Introduction

Multi-Agent Systems (MAS) that contain more than a trivial number of heterogeneous agents rely on infrastructures that support service discovery and agent interoperation. Many MASs achieve this through the use of a *Middle Agent*, such as the OAA Facilitator [10], the RETSINA Matchmaker [14] and the Infosleuth Broker [11]. These Middle Agents provide lookup services that facilitate the discovery of agents with specific capability descriptions; and may assist the communication between agents. However, agent developers often make assumptions about homogeneity across communication languages and agent interfaces. Although these assumptions may hold for smaller, closed MASs, they begin to fail as the number and diversity of agents increases, and as agents and web services appear within open environments, such as the Semantic Web[3].

Agents can be categorized as *service providers* or *service requesters*, or both, depending on their role within the MAS, or the context in which they are used. Service providers possess certain know-how or *capabilities* (e.g. planning routes between two points within a city) that service requesters may desire. In addition to this, service providers may be characterized by a set of service parameters (e.g. cost, reliability, availability etc.) and Quality-of-Service guarantees. Service requesters, on the other hand, have a set of *preferences* for particular types of parameters associated with desired capabilities. Providers advertise their capabilities and service parameters with one or more Middle Agents. Requesters submit service requests to these Middle Agents and select a provider according to their preferences.

Problems arise when a service requester has no prior knowledge of the format of the message expected by the provider or of how to interpret the response. In such cases, although service requesters may discover the desired service providers through the services of middle agents, they may not be able to communicate with them. Agent Communication Languages (ACLs) such as KQML [6], FIPA [7], and emerging web service communication languages such as SOAP [4] propose the adoption of a common ACL to which all agents adhere. Whilst such languages specify the type of communicative action that the agents perform, as well as the sender and other transport information, they do not always provide a specification of the content of the message. Other MAS such as the OAA [10] make no distinction between capability advertisements and agent queries.

The *shallow-parsing template approach* presented in this paper relaxes the constraint that agents share a common language for describing the content and format of messages. Message templates can be used in combination with advertised capability descriptions to construct and exchange messages between agents. Thus, the only assumptions made are that the agent can interact with the Middle Agent (i.e. a common Middle Agent communications protocol is used, and agents adhere to a common capability description language (CDL)[1] such as LARKS [14] or DAML-S [1]) and that it shares the same ontology as the provider. Crucially though, the two agents do not have to share the same ACL.

The contribution of this paper is twofold: we identify the assumptions that agents have to make in order to communicate with each other; and we demonstrate how a lightweight shallow parsing of the messages is sufficient to allow agents to communicate robustly.

In the following sections, we discuss how services are discovered, and how *requests*, *queries* and *responses* are constructed and interpreted, and present a method that facilitates this process. Section 2 describes the mechanism used for service discovery and capability matching within MASs that employ either

[1] It may be possible for different capability description languages (CDLs) to be used if translation services between each CDL exist (e.g. between DAML-S [1] and WSDL [5]). Typically, such translation services would be provided by the Middle Agent, and thus interoperation at this level would appear seamless to the service requester.

Matchmaker or Facilitator based Middle Agents. Section 3 describes the process of constructing messages based on message templates and capability descriptions, followed by a discussion in Section 4, and concluding in Section 5.

2 Middle Agents and Capability Matching

Middle Agents [13, 17] assist in the discovery of service providers based upon a desired service capability description. For example, Middle Agents may help service requesters locate agents that provide stock purchasing services or those that return the ticker price for a given stock. Middle Agents may mediate communication between providers and requesters [10], and support service discovery. One class of Middle Agents maintains knowledge of both the capabilities and preferences of different agents and coordinate the communication between them, thus acting as *Brokers* or *Facilitators* [10, 16]. These type of agents are often used within market-based systems. Another sub-class of Middle Agents are generally known as *Matchmakers*, *Yellow Pages* or *Directory Agent* systems [13, 14, 6, 8]. They only have knowledge about the capabilities of service providers, and do not participate in the agent-to-agent communication process. Thus, if an agent requires the services of another agent, it can query the middle agent, which then returns a list of agents whose capabilities match the request.

The behavior of a middle agent is determined by its matching process and its interaction protocols with service providers and requesters. The matching process is dependent on the capability description language (CDL) used. Infrastructures such as JINI [2] and UDDI [15] provide class-based or business-based lookup mechanisms to identify registered, distributed services. In contrast, capability-based lookup mechanisms identify agents by describing an agent or service by its functionality; i.e. by describing the service as a transformation from a set of input parameters to output parameters. For instance, an agent that reports the value of a stock on the stock market can be represented as a function that transforms the stock ticker into the stock quote.

Several CDLs exist, and vary according to their expressivity. CDLs such as WSDL [5] define parameters in terms of datatypes, and rely on matching processes that perform exact matches on parameter names etc. Larks [14] based advertisements can be matched using a multi-filtered semantic matching engine. This supports partial service matching through semantic and datatype subsumption matching. For example, requests such as *I would like an agent that can buy stocks for me* will match advertisements for agents that *can buy financial instruments*, as the matching engine can infer that *a stock is a financial instrument.* Other emerging CDLs such as DAML-S [1] are built upon semantic concepts that can be reasoned about using DAML-based First Order Logic [9].

The interaction protocol determines how agents interact with a Middle Agent, and in some cases how the Middle Agent mediates agent communication. Typically, when agents appear within a MAS, they advertise their capabilities with the Middle Agent[2]. However, the method used for discovering and interacting with a service provider can vary. The following sub-section describes how the service transaction differs for two different classes of Middle Agent: the *Matchmaker* and the *Facilitator*.

2.1 Simple Anatomy of a Service Transaction

Agents may discover available services through *Matchmakers* by constructing and submitting *requests* for services based on their preferences, e.g. a request for weather information. The format of the request is typically similar to that of the advertisement; however, instead of specifying what parameters are required to perform a service, it specifies what information the agent is willing to provide, and what information is required from the service provider[3].

The Matchmaker compares this request with advertisements maintained within its advertisement database, to determine which service providers can provide the desired service. A list of matching advertisements are then returned to the service requester, which can use them to reason about which of the service providers it will contact.

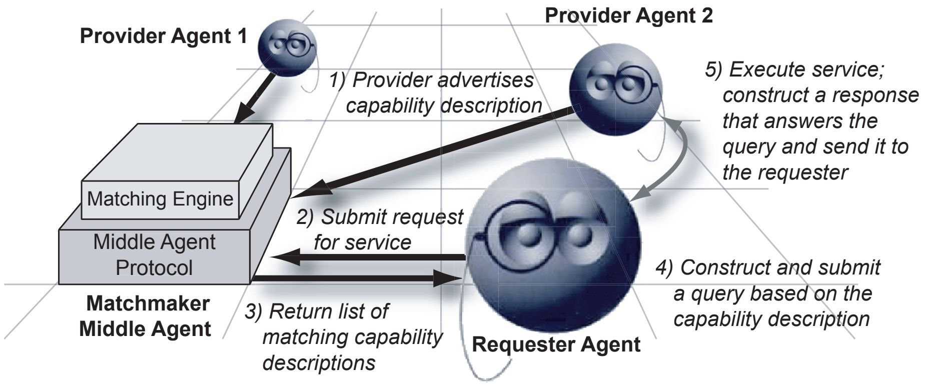

Provider Agent 1
Provider Agent 2
1) Provider advertises capability description
5) Execute service; construct a response that answers the query and send it to the requester
Matching Engine
Middle Agent Protocol
2) Submit request for service
4) Construct and submit a query based on the capability description
Matchmaker Middle Agent
3) Return list of matching capability descriptions
Requester Agent

Fig. 1. Service Discovery using a Matchmaker Middle Agent.

[2] This discussion assumes that the Middle Agents acquire knowledge of available service providers through advertisements; however, there are classes of Middle Agents, such as *Anonymizer* or *Blackboard* Middle Agents, that store only service requests or simply facilitate in the dissemination of service requests to other agents. See [13] for details.

[3] As a service requester is unable to maintain a model of all possible agent advertisements, it can never be guaranteed to provide the exact set of preferences that will match all relevant service providers. Hence a tradeoff exists between expressivity, i.e. representing current and possible knowledge, and recall, i.e. the number of possible capability matches.

After the requester has selected a provider, it constructs and submits a service *query* to the service provider. The service provider responds to the service query by sending a *reply*. This message contains the results of the service request (if such results are generated). The service transaction is complete once the service requester receives, parses, and reasons about the reply (Figure 1).

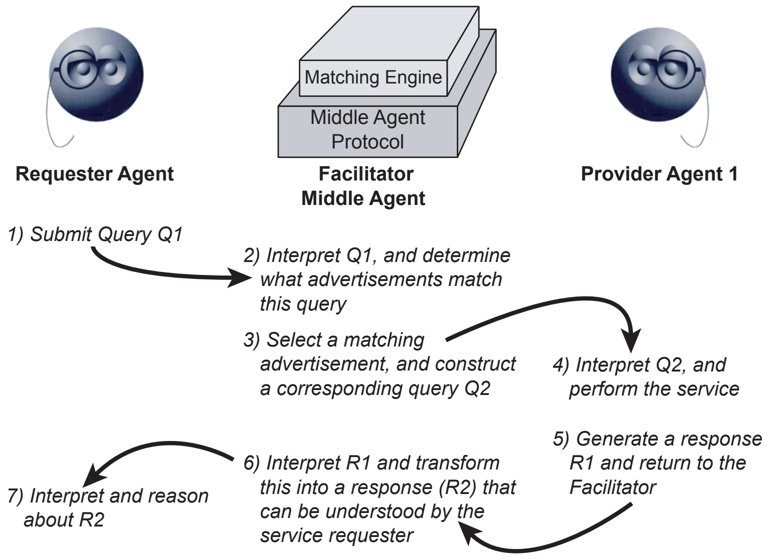

Matching Engine
Middle Agent Protocol
Requester Agent
Facilitator Middle Agent
Provider Agent 1
1) Submit Query Q1
2) Interpret Q1, and determine what advertisements match this query
3) Select a matching advertisement, and construct a corresponding query Q2
4) Interpret Q2, and perform the service
5) Generate a response R1 and return to the Facilitator
6) Interpret R1 and transform this into a response (R2) that can be understood by the service requester
7) Interpret and reason about R2

Fig. 2. Transactions between a service requester and a service provider through a Facilitator Middle Agent.

To construct a query, the service requester requires knowledge about:

1. The agent interface, i.e. the inputs required and outputs generated by the service provider. This may also include preconditions that should be satisfied before the service can be performed. These attributes typically appear within the CDL.
2. The agent communication language (ACL) used by the service provider. This includes the protocol used, information about the message header, and the format of the message content. In addition, there may be information about the attribute data types for message validation.
3. The ontology used by the service provider. The CDL specifies the format of the capability description, but knowledge of the ontology used when defining an advertisement is essential for matching and reasoning about the advertisement.

Once the requester has received an advertisement for a desired service, it can construct the query based on the agent interface defined within the advertisement, using known concepts defined within the service provider's ontology, and format the message using the same ACL as that used by the provider. Thus, in currently deployed MASs assumptions about the ontology and the ACL need to be made, if the service requester is to exchange messages with the service provider.

Transactions within Facilitator based MAS are different, in that service requesters construct service *queries* directly without a priori knowledge of the available service providers, and submit these directly to the Facilitator. In this case, the middle agent has to communicate with both the service provider and service requester, and if necessary translate between message formats. This transaction process (illustrated in Figure 2) proceeds as follows:

1. The service requester constructs and submits a query Q_1 for a desired service.
2. The facilitator interprets the query, and compares this to the advertised capability descriptions of service providers.
3. A service provider is selected, and a second query Q_2 is constructed from the contents of the first query[4]. The Facilitator then submits this query Q_2 to the service provider.
4. The service provider interprets the query Q_2 and executes the service.
5. A response, R_1, is constructed containing the results of the execution, and returned to the Facilitator.
6. The Facilitator translates R_1 into a response (R_2) that can be interpreted by the requester.
7. The service requester receives, parses and reasons about the response R_2.

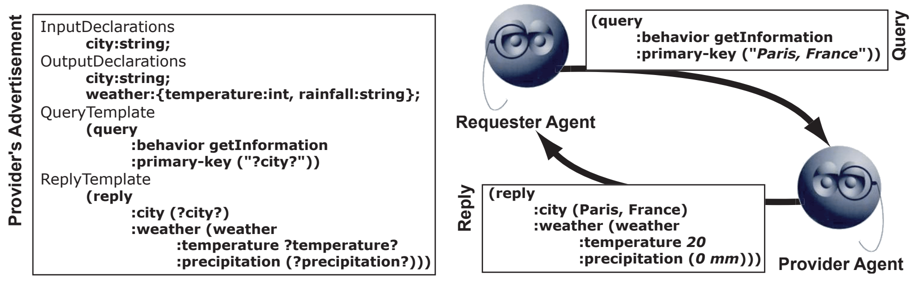
As with the Matchmaker based MAS, knowledge is required regarding the agent interface, the ACLs used by both agents, and the shared ontology. However, as communication is mediated by the Facilitator, this knowledge need only be maintained and used by the middle agent. Hence, it can mediate between two agents that do not share the same ACL (unlike a Matchmaker), and translate the messages between the two ACLs accordingly. However, knowledge is still needed about the two ACLs.

The *shallow-parsing template* approach presented here alleviates the need for knowledge about ACLs by providing preformatted message templates that describe how messages may be constructed or decomposed. The following section describes how the inclusion of these templates within the CDL facilitates communication between agents (within Matchmaker and Facilitator MASs), whilst relaxing assumptions on the ACL.

[4] Typically, facilitated systems assume a common query language and agents have a priori knowledge about the capabilities of other agents within the system. This alleviates the need for query and response translation, and thus Q_1 can be sent directly to the service provider. Likewise, the reply R_1 can be returned to the service requester without translation.

3 Query Construction and Reply Decomposition

The *shallow-parsing template* approach is based on the inclusion of preformatted message templates within the advertised capability description. These templates define the format of the messages as character sequences, and denote the location of parameter values within the message using placeholders. Two additional fields, `query` and `reply`, are inserted within the avertisement of a service provider, containing templates for query and response messages respectively.



```
InputDeclarations
        city:string;
OutputDeclarations
        city:string;
        weather:{temperature:int, rainfall:string};
QueryTemplate
        (query
                :behavior getInformation
                :primary-key ("?city?"))
ReplyTemplate
        (reply
                :city (?city?)
                :weather (weather
                        :temperature ?temperature?
                        :precipitation (?precipitation?)))
```

```
(query
        :behavior getInformation
        :primary-key ("Paris, France"))
```

```
(reply
        :city (Paris, France)
        :weather (weather
                :temperature 20
                :precipitation (0 mm)))
```

Fig. 3. Generating messages from templates.

The query template represents a sample query with unspecified input parameters. Placeholders, delimited with '?' characters, specify where input parameters should appear. The reply template can be used to identify where output parameters appear within incoming messages, so that these messages can be parsed and decomposed. As an example, Figure 3 demonstrates how the query and reply templates might be inserted into an advertisement for an agent that communicates using KQML. This advertisement describes how an agent may be queried to provide weather information about a given city. The query specifies that the reference to the city should be put between double quotes and brackets after the token `:primary-key`. For the reply, the template specifies where the desired information will be located. In the example above the agent is committed to providing temperature (i.e. 20) and precipitation (i.e. 0 mm) information, which is specified after the keys `:temperature` and `:precipitation` respectively.

The requesting agent does not have to understand the content of the query it submits, nor does it have to understand the content of the reply it receives. The only task the requesting agent has to perform is a shallow parsing of the templates to locate where the placeholders are and replace them with the required input information for the query, or to isolate and extract output information returned in the reply. For example, to compile a query that returns the current temperature in Paris, the agent simply constructs a query message from the query template by replacing the placeholder `?city?` with the name of the desired city, i.e. `Paris, France`. Again, the agent does not have to understand every detail of the reply, just the location of the information to be extracted.

By combining information about each parameter (i.e. data type, range of possible values or links to semantic concepts) with the reply template in the advertisement, the requester can decompose the response from the service provider into value-attribute tuples based on the reply template. This is achieved by comparing the template with the response, and attributing the unmatched string sequences to the corresponding parameter placeholders. For example, the reply in Figure 3 can be decomposed by comparing it with the corresponding template in Figure 3. This results in the generation of three tuples: `[city, (Paris, France)]` `[temperature, (20)]` and `[precipitation, (0 mm)]`.

This approach naturally lends itself to peer-to-peer communication within a Matchmaker based MAS. It can also be applied to a Facilitator based MAS with a slight modification:

1. *The service requester submits a query to the Facilitator.* This query contains parameters representing a subset of the requesters knowledge, and the desired preferences. These parameters can be extracted from the query by using the query template attached to the message, and using this to decompose the query into tuples.
2. *The Facilitator locates a service provider that offers the desired service.* As the matching engine is assumed to compare service requests with advertisements, the tuples could be used to construct such a service request[5]. This request is then used to select a service provider.
3. *The Facilitator constructs and submits a query to the service provider.* This is achieved by inserting the tuples into the provider's query template defined within the matching advertisement.
4. *The service provider responds by sending a reply to the Facilitator, which constructs a reply for the service requester.* The Facilitator decomposes the provider's reply into tuples, based on the message template in the provider's advertisement. These tuples can then be used to construct a reply to the requester based on the reply template sent with the initial query.

At this stage the transaction between the requester and the provider is complete and the agent has the information that it was seeking. It can now proceed to solve the problem that originally prompted the transaction.

4 Discussion

The previous section presented a simple method of constructing and decomposing messages between agents based on message templates. Although this relaxes the assumption that two communicating agents must share a common agent communication language, it does impose certain restrictions on the type of messages that can be exchanged. For example, the use of message templates establishes an upper bound on the expressivity of agent communication, by restricting the number of messages that two agents can exchange to those specified by the templates in the advertisement. In order to support a richer level of communication,

[5] It may be possible to construct several service requests based on the requester's query. However, as no formal method has yet been developed to address this, a single, arbitrary request is currently generated.

agents should agree upon an ACL based upon *social semantics* [12]. This allows agents to express messages in which unrestricted conversations can take place. However, in most cases, it is extremely difficult to construct agents that exploit this, and currently there are no agents that interoperate with each other through unconstrained communication.

The use of message templates imposes its own assumptions, namely that agents can parse and interpret the templates, and that the templates will be included within the capability descriptions. However, it is currently assumed that agents within the MAS share the same CDL, and hence can utilize a Middle Agent's discovery services. Thus, the CDL can be augmented to include optional message templates to support agent communication when there is a mismatch with assumed ACLs.

Exchanging data between two agents is only part of the solution, however. Both agents need to have some shared ontology within which the parameters that appear within the advertisement and the message templates are grounded. In addition, for all but the most trivial cases, where the vocabulary used is highly constrained, semantics are required to support the matching of advertisements. For example, an agent may advertise services for providing information about `rainfall`, and another agent may request information regarding `precipitation`. Shared ontologies are also necessary even with pre-agreed parameter vocabularies, as many concepts can be represented in different ways. For example, a city may be expressed as just the city name (e.g. `Paris`), a city and country (e.g. `Paris, France`), or with abbreviations (e.g. `Paris, Fr.`). Fortunately, formal semantics and ontology specification languages such as the DARPA Agent Markup Language (DAML) [9] are emerging, which support semantic interoperation between agents. Concepts defined in DAML can be readily combined with message templates, thus increasing the flexibility of template-based messages, and hence accessibility and interoperability of services within open environments.

5 Conclusions

For two agents to communicate successfully, knowledge about the agent interfaces, the ontologies used to define concepts, and the ACLs used to construct and parse exchanged messages is required by both participants. This paper presents a shallow parsing mechanism that provides message templates for use in message construction. Although limited to simple message exchanges, this approach relaxes the constraint of a shared ACL between agents, thus supporting flexible communication between heterogeneous agents within open multi agent systems.

6 Acknowledgments

The research was funded by the Defense Advanced Research Projects Agency as part of the DARPA Agent Markup Language (DAML) program under Air

Force Research Laboratory contract F30601-00-2-0592 to Carnegie Mellon University. Special thanks goes to Martin van Velsen for his contribution towards the development of many of the ideas presented in this paper.

References

1. A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, and H. Zeng. Daml-s: Semantic markup for web services. In *International Semantic Web Working Symposium*, 2001.
2. K. Arnold, B. O'Sullivan, R. Scheifler, J. Waldo, and A. Wollrath. *The Jini Specification*. Menlo Park, CA:Addison Wesley, 1999.
3. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.
4. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, 2000.
5. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.
6. T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an agent communication language. In *Proceedings of the third Conference on Information and Knowledge Management, CIKM94*. ACM Press: New York, 1994.
7. FIPA. Foundation for Intelligent Physical Agents. http://www.fipa.org/, 1997.
8. M. Genesereth and S. Katchpel. Software Agents. *Communications of the ACM*, 37(7):48–53, 1994.
9. J. Hendler and D. L. McGuinness. Darpa agent markup language. *IEEE Intelligent Systems*, 15(6):72–73, 2001.
10. 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.
11. M. Nodine and A. Unruh. Facilitating Open Communication in Agent Systems. In M. Singh, A. Rao, and M. Wooldridge, editors, *Intelligent Agents IV: Agent Theories, Architectures, and Languages*, pages 281–296. Springer-Verlag, 1998.
12. M. P. Singh. A social semantics for agent communication languages. Technical Report TR-99-03, 22, 1999.
13. K. Sycara, K. Decker, and M. Williamson. Middle-agents for the internet. In *Proceedings of IJCAI-97*, January 1997.
14. K. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record (ACM Special Interests Group on Management of Data)*, 28(1):47–53, March 1999.
15. UDDI. The UDDI Technical White Paper. http://www.uddi.org/, 2000.
16. M. Wellman. A Market-Oriented Programming Environment and its Application to Distributed Multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1:1–23, 1993.
17. H. C. Wong and K. Sycara. A taxonomy of middle-agents for the internet. In *Proceedings of the Fourth International Conference on MultiAgent Systems*, pages 465 – 466, July 2000.