Designing and Implementing a Multi-Agent Architecture for Business Process Management*

Timothy J. Norman* Nick R. Jennings* Peyman Faratin* E. H. Mamdani†

* Department of Electronic Engineering, Queen Mary and Wesfield College, University of London, Mile End Road, London E1 4NS, UK.
{T.J.Norman,N.R.Jennings,P.Faratin}@qmw.ac.uk

† Department of Electrical and Electronic Engineering, Imperial College of Science, Technology and Medicine, University of London, London SW7 2AZ, UK.
e.mamdani@ic.ac.uk

Abstract. This paper presents a general multi-agent architecture for the management of business processes, and an agent design that has been implemented within such a system. The autonomy of the agents involved in the system is considered paramount. Therefore, for agents to agree on the distribution of problem solving effort within the system they must negotiate. The knowledge sharing and negotiation functions of such an agent are focused on in this paper.

1 Introduction

The typical modern commercial enterprise consists of a number of, possibly physically distributed, semi-autonomous units, each with a degree of control over local resources or with different information requirements. These semi-autonomous units of a single or a number of collaborating organisations are coordinated via a "business process". This business process specifies the tasks that must be performed and the decisions to be made in the generation of a product or service. The motivation behind the ADEPT (Advanced Decision Environment for Process Tasks) project is that an agent based approach should be suitable for implementing systems to manage business processes. (An example of a business process managed by an ADEPT system is described by Jennings *et al.* [10].)

The ADEPT multi-agent architecture consists of a number of autonomous agencies. A single agency consists of a, possibly empty, set of subsidiary agencies represented by a single responsible agent. Thus, the architecture may model a hierarchical or flat organisational structure, or a mixture of the two. The responsible agent may be approached by other autonomous agents for the provision of a "service" (i.e. some unit of problem solving activity). A service corresponds to either an atomic task within the business process, or a composition of a number of other services from various agents (see section 2.1). Each agent acting autonomously will constantly assess the situation and decide how to

* This paper reports on aspects of the ADEPT project. ADEPT is a collaborative project under the DTI/EPSRC Intelligent Systems Integration Programme (ISIP). The project partners are BT Laboratories, ICI Engineering Technology, Loughborough University, and Queen Mary and Westfield College.

commit the resources of its agency, whether to call for services from other agents based on some prior agreement, or negotiate for new “service level agreements”.

Thus, the ADEPT system and agent architectures (see sections 2 and 3 respectively) are designed to ensure maximum flexibility to adapt as the business process changes. The autonomy of each agent and the agreements it enters into with other agents is the key to this flexibility. An agent will behave in a pro-active manner where possible; for example it can enter into negotiation in anticipation of a future need for some service. Agents use negotiation as a mechanism for the distribution of resources, problem solving effort, and the dependencies between the activities to be carried out. The result of successful negotiation is a service level agreement. The existence of such an agreement is both a precondition for a service to be provided, and serves to bind future activity and any subsequent resource commitments.

ADEPT provides a method for designing agent-oriented business process management system, an agent implementation that is suitable for operation within such a system, and a technology for solving the problem of integrating an enterprise in the performance of a business process. This paper addresses the first of these aspects in detail and introduces the agent implementation; Jennings *et al.* [10] presents the implementation in greater detail and the use of ADEPT as a solution technology. Section 2 describes the ADEPT system architecture and the constraints that this design places on agent design. These design constraints motivate the agent implementation presented in section 3. Section 4 serves to conclude and compare the ADEPT architecture with other multi-agent architectures designed for similar purposes.

2 The ADEPT Multi-Agent Architecture

Designing a multi-agent system to manage business processes involves the principled transformation of some description of that business process into a number of communicating and cooperating software agents. This is different from business process re-engineering (or modification). An agent-oriented business process management system is designed to enhance the operation of an existing business process rather than to modify it, although the business process may be modified before such a system is built.

The resulting system serves to integrate the geographically and/or logically distinct sections of an organisation (or a number of organisations) through communication. This enables more effective coordination between these existing sections in the operation of a business process. The ADEPT architecture has the capability to model the structure of flat or hierarchical organisations, or those structured using a mixture of the two through the concepts of agents and agencies described in section 2.1. The architecture attempts to reflect the structure of an organisation, and the business process is used by the agents as a specification of how services are performed. Furthermore, the use of distributed, autonomous agents as an implementation means that, if the business process is changed, minimum changes to the business process management system will be necessary. In contrast, existing business process management systems rigidly adhere to the operation of a particular business process, if even minor changes are made to the business process, the whole management system may need to be re-written. This section describes the principles guiding the design of an ADEPT system, and the minimum requirements for a particular agent design if it is to successfully participate in such a system.

2.1 Agents and Agencies

An ADEPT system may consist of a mixture of hierarchies of agencies and peer structures enabling many organisational structures to be reflected in the multi-agent system. The design of such a system involves the transformation of some business process description into a number of agencies, each with a connection to some common communication medium (see figure 1), although in practice feedback from the multi-agent system may motivate re-engineering of the business process. Agents may communicate via Email, or an Object Request Broker [16], for example.

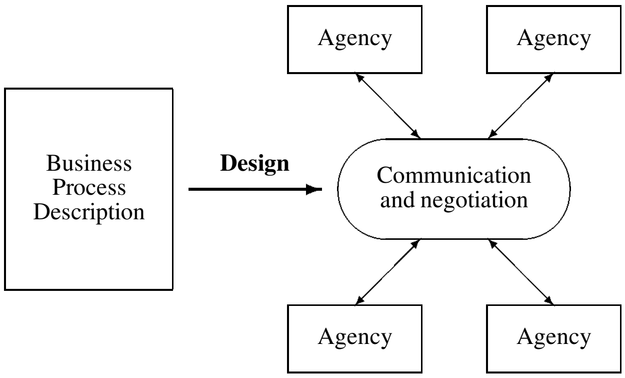

Fig. 1. Designing an agent-based business process management system.

An agency is recursively defined: an agency consists of a single responsible (or controlling) agent, a, possible empty, set of tasks that the responsible agent can perform, and a, possibly empty, set of sub-agencies (see figure 2).[2] For example, agency D has a single responsible agent that has two distinct tasks (TD1 and TD2) and three sub-agencies (agencies E, F and G). (A task is an atomic service; a service being some unit of problem solving activity.) The responsible agent represents the interests of the agency to its peers.[3] Any communication with an agency must go through the responsible agent. A sub-agency (e.g. agency G is a sub-agency of agency D, figure 2) typically behaves in a cooperative manner towards its responsible agent, this agent being responsible for representing the interests of the agency in the wider community. This relationship between sub-agency and responsible agent can be viewed as a type of social commitment, and provides a mechanism for the encapsulation and abstraction of services (see below). Suppose that the responsible agent requests a service from one of its sub-agencies.

[2] An agency must contain at least one task or two sub-agencies for it to be meaningful.

[3] Peer agencies are those with responsible agents that may communicate without crossing an agency boundary. For example, in figure 2 agency F is a peer of agency E and agency A is a peer of agency D, but agency E is not a peer of agency A.

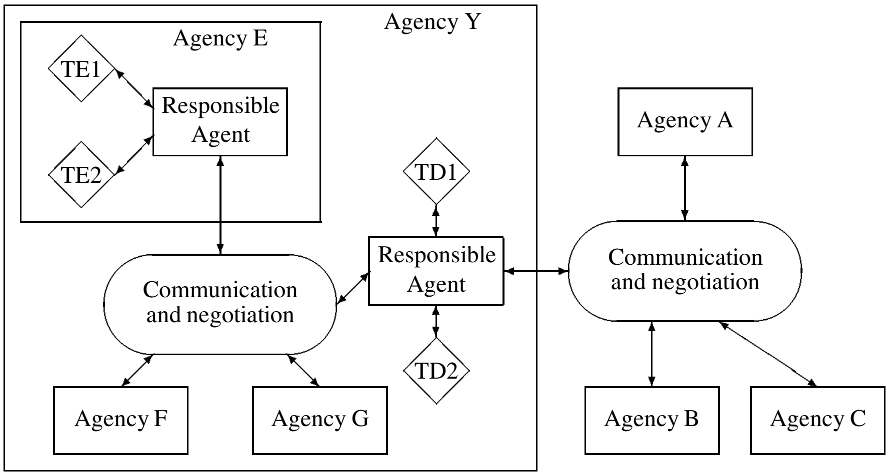

Fig. 2. The logical hierarchy of agencies.

This request cannot be flatly refused, a legitimate reason must be given. However, a sub-agency is not a subroutine, but rather a subsidiary agency. A sub-agency retains a high degree of local autonomy, but will necessarily cooperate with the responsible agent where possible. For example, the manager of a design department may request a design engineer to work on a particular project. If that engineer is able to perform that task within certain constraints (e.g. to meet a deadline), the request will be accepted, but the conditions under which the request is met may be open to negotiation. In contrast, the relationship between peer agents is more open; an agent is not necessarily under any obligation to accept a request from its peer. However, an agent will come to a mutually acceptable agreement with a peer agent if it is in its best interests to do so, but may take into account whether the peer is a member of the same organisation, or whether they are negotiating for an out-sourced service. A responsible agent provides a specification of the services that it can and is willing to provide to its peers, even though some of the activity will require the assistance of its sub-agents.

The hierarchical structure of agencies in the ADEPT environment provides a mechanism for the encapsulation and abstraction of services. As an example, consider the agency illustrated in figure 2. Suppose that this diagram represents the structure of an organisation in which the design department is the agency that is expanded. In this department, the responsible agent represents the manager of this department (i.e. the agent through which other departments may contact the design department), sub-agency W represents a single design engineer that is capable of performing two distinct tasks (or atomic services), and one of the other agencies represents a team of surveyors. Furthermore, suppose that the department manager has registered a "cost and design network" service which can be provided by the design department to other agencies in the organisation. Before the department manager is able to register this service, it must know that it is able to provide that service to other agents in the community under certain conditions.

Suppose that for the manager to be able to provide a "cost and design network" service, the design engineer must be able to provide a "design network" service. Also, the engineer must collaborate with a surveyor to ensure that a design that is proposed will be consistent with the geographical requirements of the proposed network site. Therefore, for the manager to register this "cost and design network" service, at least one design engineer must register with its peers and responsible agent (if one exists) the service "design network", and the agent representing the team of surveyors must register a "survey site" service. Then, subject to a negotiated contract, the department manager may agree to cost and design a proposed network installation with certain characteristics at a particular location for another agent.[4] Note that it is neither necessary for the agent requiring a "cost and design network" service to know how this is achieved, nor is it necessary for the department manager to know how to design a network or survey the geographical requirements of a particular site. This provides a mechanism for agents to represent and reason about services at an appropriate level of abstraction.

To the agent managing a task, that task can be viewed as an atomic service with well defined input, output and functional specifications. In general, the services that an agent registers in the community are the tasks that it is able to perform plus services constructed through the combination of its tasks and services available from its sub-agencies. (Although, in unusual circumstances an agent can use services provided by its peers in combination with other services to construct a new service.) However, during its lifetime an agent may register new services as they become available, or withdraw services if necessary (e.g. due to other agents or tasks becoming inactive). An agent will agree to provide a service that it has registered only if a mutually advantageous agreement can be made with the agent requesting that service. The agent registering a service is responsible for negotiating the terms under which that service may be provided to other agents, and the reliable execution of the service under the terms of such an agreement. If an agent agrees to provide a service to another, it is referred to as the "server" agent, and the agent receiving the service is the "client" agent for that service (roles that may be reversed). (Note that this characterisation of agents as client agents and server agents is different from the use of these terms in more traditional client-server systems [5]; the agents are autonomous, and will only cooperate under a negotiated agreement.)

It is possible to design a system whereby more than one agent may act as the responsible agent for different purposes, or the roles of the agents within the agency may change. The simple structures proposed here are common in existing organisations, and hence enable many organisations to be modeled within the ADEPT architecture. However, more work is required in analysing the impact of different organisational structures and the migration of sub-agencies from one agency to another.

2.2 Inter-Agent Communication

To enable functions such as negotiation, each agent must have the ability to communicate with agents that it needs to interact with through some mechanism. However, agents

[4] A negotiated contract for the "design network" service may be required before the "cost and design network" service is offered, or this may be arranged at run time. This is the choice of the system designer.

may be developed for different purposes, by different people at different times, and so neither the languages used in the development of the agent software nor the methods of representing knowledge in each agent will be uniform; i.e. agents are heterogeneous. In common with a number of existing approaches to software agent inter-operation (e.g. the SHADE and PACT projects [24] and the Knowledgeable Community project [23]), for an agent to participate in an ADEPT environment it is necessary for it to communicate using a common expressive language. This common language consists of a protocol (i.e. a set of related speech acts [1, 20]) and a syntax for expressing information. Furthermore, for one agent to understand the meaning of another's communicative actions they must either share a common information model[5] or have the ability to transform information between their respective models. This use of a common knowledge sharing language and a common semantic interpretation of the symbols used within messages composed using this language enables agents within the community to: (1) know the *intention* of a communicating agent via a set of speech acts (e.g. is the message intended to inform, deny, or request some proposition?); (2) *interpret* the contents of the message through the use of a common syntax (e.g. the Knowledge Interchange Format, KIF, which is a prefix version of first order predicate calculus with certain extensions designed for this purpose [6]); and (3) *understand* what the contents of the message mean via the use of ontological commitments (e.g. what is the meaning of the proposition that the agent is being informed of, that is being denied, or requested). These three aspects of agent communication (expanded on below) enable agents to communicate their beliefs, goals and other mental states, and thereby negotiate and perform other complex communicative functions.

An agent operating in an ADEPT environment must use one of a set of specified communicative action types to specify the *intention* of a message (cf. Barbuceanu and Fox [2, 3], and Cohen and Levesque [4]). Consider the action "propose". This indicates that the agent sending the message intends the contents of the message to be interpreted as a proposal for the provision of a particular service that the recipient has registered as being able to provide (see section 3.1). Suppose that the manager of a design department has a prior agreement to provide a peer agent with a "cost and design network" service. On the basis of this contract, the manager receives a message invoking that service. Furthermore, suppose that the pre-arranged "design network" service fails. In response to this failure of prior arrangements, the manager must now negotiate with some other design engineer for this "design network" service. During negotiation, a number of proposals and counter-proposals are passed back and forth until the manger makes an agreement with a designer.[6]

As well as understanding the intention behind the message, a recipient must be able to *interpret* the contents of a message for it to be understood. Therefore, either each agent must use the same syntax to communicate information, or the syntax used must be ex-

[5] An information model, or ontology, is a data structure that provides a context for the interpretation of symbols. The term ontology has been defined in this context by Gruber [8] as a "specification of a conceptualisation".

[6] If no agreement can be made, the manger must refer back to the agent with which it has an agreement to provide a "cost and design network" service. For the future, we are investigating techniques for dynamically revising the business process in such cases.

plicitly stated in the message (e.g. Prolog or KIF). In the latter case, for agents to communicate they must both be able to interpret the syntax specified in the message. For example, if agent `x` wishes to communicate to agent `y` that it believes `p`, the expression contained in the message may be `believes-that(x, p)` or `x believes-that p`. The former using a prefix notation and the latter an infix notation for the operator `believes-that`. For agent `y` to be able to interpret the contents of a message from `x` they must agree on a common syntax for the representation of communicated knowledge.

The final requirement for effective inter-agent communication is that the agent receiving a message must be able to *understand* the intended meaning of the symbols contained in the message. Suppose that the agent managing a design department has an agreement with the team of surveyors in that department to survey particular locations when requested. Then if the manager asks the team to survey a location, the agent representing that team of surveyors must know what is meant by a location for it to understand the message. The manager of a design department may interpret the symbol "location" to refer to the postal address of the client requiring the network and the symbol "site" to refer to the location of the proposed network. The design engineer may interpret the symbol "location" to be the grid reference of the network site. These agents do not share the same model of information. For these agents to communicate, they must share a common interpretation of the message contents.

2.3 Inter-Agent Negotiation

In the ADEPT environment agents are autonomous; i.e. agents have control over the tasks that they may perform, the resources available to them and how they coordinate their activities with other agents. Therefore, the only way in which agents may cooperate in solving problems in such environments is through negotiation. Negotiation is defined as the process of joint decision making where parties verbalise their (possibly contradictory) demands and then move towards agreement [17]. The progress towards an agreement depends on the *strategies* adopted by the agents involved, which may be different (see section 3.2). Furthermore, in a practical system these agreements must be made in a timely manner, avoiding unnecessary communication (especially in domains such as the Internet where bandwidth is at a premium). Hence, there are three primary motivating factors in the development of inter-agent negotiation mechanisms: (1) to minimise computational effort in generating a response; (2) to minimise communication overheads; and (3) to retain as much as possible the autonomy of each agent involved in the negotiation. In general, the convergence of the negotiation process (and hence the volume of communication required) depends on the relationships between the parties involved, and the negotiation strategies employed.

The contract net protocol [22] is often used in the coordination of agents in cooperative problem solving, although it does not strictly involve negotiation. The protocol is initiated by a manager agent that breaks the problem into a number of sub-problems and then searches for contractor agents that can solve each sub-problem in isolation. An agent that is able and willing to solve a sub-problem will table a bid with the manager agent. These bids are then evaluated and the contract is offered to the agent that tabled the best bid. Once contracts are generated for every sub-problem, the sub-solutions subsequently received are combined to produce a solution to the original problem. This protocol does not strictly involve negotiation (according to our definition) because bidding agents have only one shot at the available sub-problem. Richer models that address this criticism are the recursive negotiation model [14] and the sequential decision-making model [25]. In the recursive negotiation model [14], agents exchange information and plans, and produce critiques of other's proposed plans in an attempt to find a mutually acceptable solution to the distribution of action and control in solving the problem. However, each agent is required to be entirely cooperative (and hence not entirely autonomous), the focus being on the resolution of mismatch in the cooperation strategy. In the ADEPT environment agents require a richer model of negotiation (involving both cooperative and uncooperative strategies) that enables them to retain their local autonomy.

For agents to negotiate in the ADEPT environment, they require: (1) a protocol; (2) a data structure that represents the result of negotiation (i.e. something to negotiate over); and (3) a reasoning model. The protocol is based on the performative definitions described in the inter-agent language (see section 2.2). The result of a successful process of negotiation is an agreement (referred to as a service level agreement, or SLA) to provide some service under terms and conditions which are acceptable to all parties. Agents negotiate over the contents of an SLA, and if the agents involved in negotiation agree on a final SLA this represents a binding contract between these agents. The SLA is the standard unit of exchange in the ADEPT environment, and hence serves to focus the negotiation process. When negotiating over the contents of an SLA, agents must not only agree how a problem is to be solved among the participating agents, but also under what conditions these problem solving activities should take place; the structure of an SLA may vary between implementations, but an example is briefly described in section 3.2 (see Jennings *et al.* [10] for more detail). The reasoning model used will depend on a particular agent implementation, but see section 3.2 for a discussion.

3 An Agent Implementation for the ADEPT Environment

The ADEPT agent implementation described in this section consists of a number of functional components responsible for each of the agent's main activities: communication, negotiation, execution of services, and situation assessment (see figure 3). This agent architecture is broadly based on the GRATE [9] and ARCHON [11] agent models. A multi-agent system involving 9 agents, 10 services and 93 tasks has been designed and implemented to manage a British Telecom business process (described in more detail by Jennings *et al.* [10]). The agents are implemented using a modified version of the C Language Integrated Production System (CLIPS) [7], and communicate via the DAIS platform (a commercial implementation of the CORBA specification [16]).

DAIS distributes messages via its Object Request Broker (ORB) transparently between registered objects. The ORB receives requests from a "client" (i.e. an agent or task) to send a message to an "object" (i.e. another agent or task). The broker locates the object referred to by the client and delivers the message to that object. DAIS provides an interface to this brokering service through which the registration and deregistration

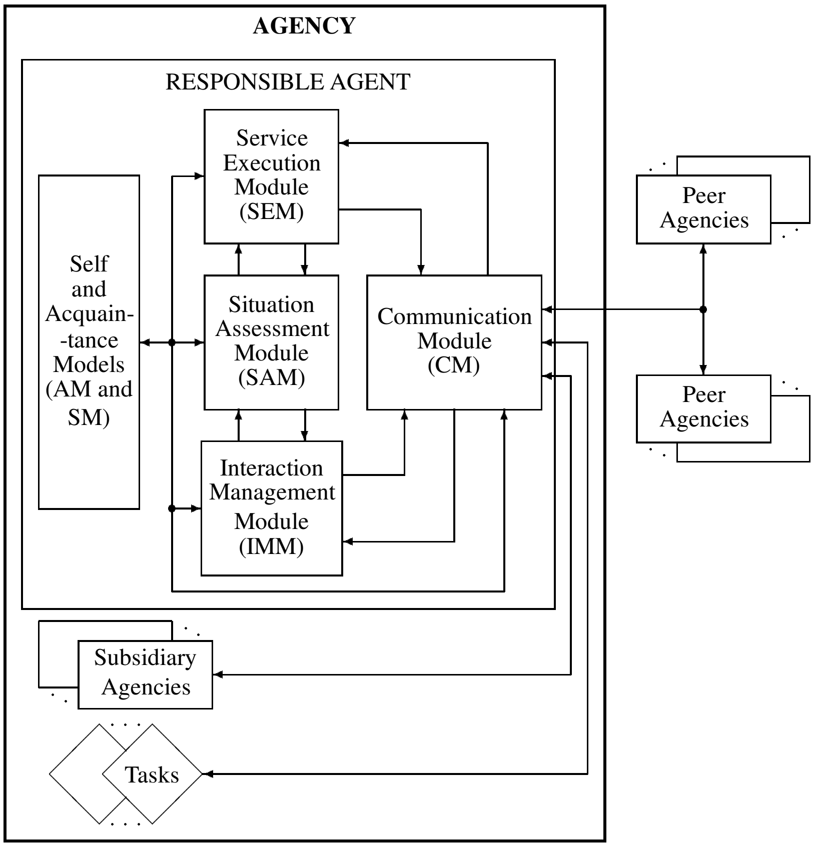

Fig. 3. ADEPT agent architecture.

of agents and tasks and message requests are handled.

An agent models the services that it can provide, the resources available to it, its current schedule of activity and other self knowledge in its self model (SM). The existence and known capabilities of other agents, along with histories of past encounters with them, are stored in the agent's acquaintance model (AM). Aspects of these models are used by the various functional components of the agent. The situation assessment module (SAM) maintains the agent's schedule of activity. It uses information concerning requests for services by other agents, services that it has agreed to provide for other agents, failed tasks and other relevant information to guide negotiation for and execution of services via the interaction management and service execution modules respectively. The service execution module (SEM) is responsible for the initiation and management of the services that the agent has agreed to provide to other agents, and for the invocation of services provided to it by other agents. If a service fails, the SEM may attempt to execute it again, or refer back to the SAM for rescheduling of, or renegotiation for that service. When prompted by the SAM, the interaction management module (IMM) will negotiate (or renegotiate) for the service required. The IMM will also, referring back to the SAM for scheduling information, represent the interests of that agency when another agent requests the provision of a service registered by the agent. Both the SEM and IMM communicate with other agents and with the tasks managed by that agent through the communications module (CM). The CM is responsible for packaging messages in a form that can be understood by the intended recipient of that message, and the receipt, interpretation and forwarding of messages received by other agents and tasks. The remainder of this section focuses on the provision of services (i.e. negotiating for services) through the communication and interaction management functions of this agent architecture.

3.1 Communication

The management of communication between agents involves the possible transformation of content, packaging of that content in an appropriate form, and the routing of the resulting message to the intended recipient. The DAIS ORB provides transparent routing of messages to registered agents or tasks by name. Therefore, it is the responsibility of the communications module to generate message packages that can be transmitted through DAIS.

Consider a negotiation message. Suppose that the IMM instructs the CM to send a proposal to an agent that is capable of providing a particular service and is willing to negotiate. The agent intends to ask the question "Will you provide this service under these conditions?". The CM uses its acquaintance model to determine how to go about presenting this proposal to the other agent in a form that it can understand. According to the performatives available to an ADEPT agent, the CM sends an appropriately structured propose message to the agent concerned. If the CM fails to contact the intended recipient, this failure is reported back to the IMM. For example, the CM may not be able to translate the contents of the message into a form that can be understood by the recipient agent.

At present, negotiating agents use the six communicative actions: propose, counter-propose, accept, reject, confirm and deny (cf. Barbuceanu and Fox [2] and Mayfield *et al.* [15]). A propose message is an initial proposal for the provision of a specified service, a counter-propose message is a reply to the previous proposal made by the other agent, and either agent may accept or reject the other's (counter)proposal. A reject terminates the negotiation process, and an accept requires either a confirmation or a denial from the other for the agents to become committed (or not in the case of a deny message) to the (counter)proposed agreement. Each of these message types are directed towards specific agents determined by the IMM (see section 3.2 for more detail on the negotiation process). The task of the CM is to ensure that the message is delivered to the correct agent in a form that is acceptable to that agent.

An inter-agent message is a package that contains: one of a limited number of communicative actions, the identity of the sender, recipient and strand of conversation (via the conversation identifier), the service being negotiated for, and the information model with reference to which the contents of the message should be understood (see figure 4).

```
(message
  (action: <communicative-act>)
  (sender: <agent-id>)
  (recipient: <agent-id>)
  (conversation: <conversation-id>)
  (service: <service-name>)
  (info-model: <model-id>)
  (content: <expression>)
  )
```

Fig. 4. The general structure of an inter-agent message.

All messages passed between agents in the ADEPT environment conform to this structure.

The CM has the responsibility of ensuring that conversations are kept distinct despite the fact that multiple conversation strands for multiple instances of the same service may be pursued concurrently. To distinguish between the various conversations for the same service and for different services, agents specify the sender, recipient and conversation identity and the service name in the message.

At present the actions used have no formally specified semantics [4], but such a specification is presently under development. This field of an inter-agent message specifies the action that the agent is performing by sending the message. For example, the agent may be proposing, counter-proposing, accepting, rejecting, confirming or denying if it is in the process of negotiating with another for the provision of a service.

The “info-model” field refers to the semantics of the expression contained in the “content” field. This enables an ADEPT programmer to write agents that are able to communicate using different information models. The agent sending the message will use its acquaintance model to determine the information models that can be interpreted by the intended recipient of the message to ensure that the message can be understood. This field is also used as a heuristic by the recipient of a message to decide whether or not the message is acceptable. If the agent cannot understand a message in terms of the information model specified an appropriate reply is generated without considering the contents of that message. The content of a message is an expression in the syntax of the inter-agent language which may be understood with reference to the information model specified.

3.2 Negotiation

At present, ADEPT agents negotiate for SLAs that are characterised by their scheduling attributes, meta-service details, inputs and outputs. For example, for agent y to agree to provide agent x with service s then they must agree a schedule for the availability of s, the meta-services details (e.g. price or frequency of invocation), what inputs are required for y to provide service s, and what outputs are returned to x. The scheduling attributes of s include how long it should take to complete (duration) and between what times should the service be available to x (start time and end time respectively). Thus, x and y must not only agree that y will do s for x, but also agree a time frame that does not conflict with either agent's prior plans. The meta-service details of s include the volume of service invocations that are permitted between the start time and end time, the price paid per invocation of s, and the penalty that y will incur for every violation of the agreement. Finally, x and y must agree on the information that x provides to y when invoking s, and the information that is returned to x from y on completion of the service. See Jennings *et al.* [10] for more detail.

Each agent requires a model of negotiation to capture the richness of interaction that can take place in the ADEPT environment. This requirement has lead to the development of a deep model of goal-directed negotiation, with which agents can agree the distribution of effort in the satisfaction of these goals. There are three principles guiding the design of this model. First, the model should sufficiently abstract for it to be applicable to a wide variety of problem domains. Second, there may be domains where the model does not fully guide decision making, and so it should be flexible to extension. Finally, agents are computationally bounded, and so they require tractable decision-making processes.

Existing formal models of agent coordination and negotiation provide valuable theoretical insights, but in general, rely on unreasonable assumptions. For example, the contract net protocol fails to capture the recursive nature of negotiation where agents relax local requirements to achieve an acceptable overall solution. In contrast, game theoretic models of negotiation (e.g. Rosenschein and Zlotkin [19]) make assumptions that are invalid in real domains; for example, agents are assumed to have no computational limitations, have completely specified preferences, and complete knowledge about the problem domain. However, these methods are not suitable for negotiation in open and unpredictable environments. The aim of the ADEPT negotiation model is to develop computationally tractable algorithms for more complex inter-agent negotiation in real domains.

The ADEPT model employs both declarative and procedural knowledge. The declarative knowledge is represented as a causal network, and explicitly models what is being negotiated for and why the negotiation is taking place (i.e. it sets the negotiation context). For example, the agent may be negotiating over the price of a service because it believes that it is being charged too much. Similarly, an agent may need to negotiate over the start time if the time proposed by the other party conflicts with an existing commitment.

The procedural knowledge base, represented as a set of strategies and mechanisms for selecting between them, specifies which actions should be taken given the declarative knowledge. For example, given that the agent needs to negotiate over price, the knowledge base may indicate that Boulware is a good strategy to adopt if the agent has a long time to reach an agreement, or if there are many suppliers of the service. (Boulwarism is a strategy in which the negotiator makes a reasonable initial offer and then sticks firm throughout the negotiation [18].) In such cases the agent generates a price offer and continues to counter-propose that initial offer throughout the negotiation. Alternatively, if the agent wants to reach an agreement for a scarce service or if it is a sub-agent negotiating with the responsible agent of its agency (ref. section 2.1), then it may adopt the more cooperative tit-for-tat strategy. This means that an agent will make concessions when the other concedes and stands firm when the other agent is uncompromising. As the negotiation progresses an agent may move between strategies as circumstances change.

4 Conclusion and Related Work

The ADEPT multi-agent architecture is presented as a novel solution to the problem of software agent inter-operation in domains such as business process management. The architecture has the capability to model the structure of hierarchical or flat organisational structure, or a mixture of the two through the concepts of agents and agencies described in section 2.1 (these are common structures in existing organisations). In coordinating the action of agents within a multi-agent architecture it is important to find a balance between the autonomy of agents within the system and the communication overheads involved in coordinating action. Agents with little autonomy typically require less communication bandwidth; for example, a subservient agent will simply follow instructions. Agents with greater autonomy must be persuaded to act on another's behalf, and hence agents must negotiate for services. The ADEPT architecture supports the encapsulation of services through the hierarchy of agencies, and so enables abstracted services to be negotiated for, reducing communication overheads. To enable service encapsulation, subsidiary agencies behave more cooperatively with the responsible agent of their agency (see section 2.1), surrendering a degree of autonomy. However, these agents retain control over their own resources, the tasks that they perform and their coordination and communication with other agents. They simply cooperate in negotiation with their responsible agent where possible; i.e. they are subsidiary, not subservient. Peer agents have no such social commitment, and so the provision of a service is predicated on there being a mutually acceptable agreement produced through negotiation. However, an agent may be more cooperative with a peer that represents a different department of the same organisation than a peer representing the interests of a different organisation. This paper has stressed the need for individual agents to retain autonomy of resources, the tasks they can perform and coordination with other agents. Therefore, negotiation plays a central role in the interaction of agents within an ADEPT system, and an expressive common language is required to enable agents to negotiate.

The federation architecture [12, 21, 24] provides a different method for the organisation of a multi-agent system in similar domains. Agents are organised into groups, each group being associated with a single "facilitator" to which an agent surrenders a degree of autonomy. A facilitator serves to identify agents that join or leave the system and enables direct communication with other agents regardless of their location by the facilitator setting up a direct link; functions that are somewhat equivalent to the DAIS ORB [16]. In addition, the facilitator provides anonymous communication (i.e. agents are informed of events in which they are interested without reference to the original sender), translation of message content between different information models, problem decomposition and distribution of sub- problems to agents unspecified by the original sender and delayed communications in the event of an agent being temporarily off-line. (The architecture proposed by the Knowledgeable Community project [23] is similar in structure, but these functions are distributed between a "facilitator", "mediator" and "ontology server".) The federation architecture enables agents to communicate without concern for the particular syntactical and semantic requirements of the recipient. An agent may also send a message without specifying the recipient; the content-based routing of these messages being performed by a facilitator. However, this content-based routing of inter-agent messages removes the agent's control over who receives a particular message. During negotiation, agents require secure communication between the participants. Although the federation architecture supports direct communication, other functions such as reporting events that are of interest to a third party that are supported by a facilitator must be suppressed in such cases. Therefore, it may be necessary for facilitators to make valued judgments about whether or not a particular event should be reported and whether this conflicts with other interests it is representing. At present the federation architecture has only been used for the inter-operation of purely cooperative agents (e.g. SHADE and PACT [24]). Note that agent inter-operation through cooperation only is not a good model if the business process involves more than one organisation. An additional difficulty with the federation architecture is that it does not support the encapsulation of services. The ability to model both peer and hierarchical structures in ADEPT is founded on organisational models where an enterprise is logically divided into a collection of services. The agent-agency concept in ADEPT draws on this principle to group services within the system where it makes pragmatic sense; a flexibility that is not available in the federation architecture.

Future work in the ADEPT project includes the extension and more rigorous specification of the speech acts used in negotiation. This will enable an ADEPT agent to have a more fine control over its negotiation strategy, and hence lead to the development of a richer model of negotiation. There are few evaluative studies of negotiation, and most of these focus on the effects of different negotiation strategies on the agent society [13]. The ADEPT environment is being used as a test-bed for the empirical evaluation of various negotiation strategies, and methods of changing between strategies during negotiation as more information becomes available.

References

1. J. L. Austin. *How to do things with words*. Harvard University Press, 1962.
2. M. Barbuceanu and M. S. Fox. Cool: A language for describing coordination in multi-agent systems. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 17–24. AAAI/MIT Press, 1995.
3. M. Barbuceanu and M. S. Fox. The design of a coordination language for multi-agent systems. In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III — Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1996. In this volume.
4. P. R. Cohen and H. J. Levesque. Communicative actions for artificial agents. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 65–72. AAAI/MIT Press, 1995.
5. G. F. Coulouris and J. Dollimore. *Distributed systems: Concepts and design*. Addison Wesley, 1988.
6. M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.
7. J. C. Giarratano and G. Riley. *Expert systems: Principles and programming*. PWS Publishers, 2nd edition, 1994.

8. T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.
9. N. R. Jennings. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence*, 75(2):195–240, 1995.
10. N. R. Jennings, P. Faratin, M. J. Johnson, P. O'Brien, and M. E. Wiegand. Using intelligent agents to manage buisness processes. In B. Crabtree and N. R. Jennings, editors, *Proceedings of the First International Conference on Practical Applications of Intelligent Agents and Multi-Agent Technology (PAAM96)*, pages 345–360. Practical Applications Company Ltd., 1996.
11. N. R. Jennings, E. H. Mamdani, I. Laresgoiti, J. Perez, and J. Corera. Using ARCHON to develop real-world DAI applications for electricity transportation management and particle accelerator control. *IEEE Expert*, Dec 1996.
12. T. Khedro and M. R. Genesereth. The federation architecture for interoperable agent-based concurrent engineering systems. *International Journal on Concurrent Engineering, Research and Applications*, 2:125–131, 1994.
13. A. P. Kosoresow. On the efficiency of agent based systems. In Groen, Hirose, and Thore, editors, *Proceedings of Intelligent Autonomous Systems 3*, pages 551–560, 1993.
14. B. Lâasri, H. Lâasri, S. Lander, and V. Lesser. A generic model for intelligent negotiating agents. *International Journal of Intelligent and Cooperative Information Systems*, 1(2):291–317, 1992.
15. J. Mayfield, Y. Labrou, and T. Finin. Evaluation of kqml as an agent communication language. In M. Wooldridge, J. P. Müller, and M. Tambe, editors, *ntelligent Agents II: Proceedings of the IJCAI-95 workshop on Agent Theories, Architectures, and Languages*, volume 1037 of *Lecture Notes in Artificial Intelligence*, pages 347–360. Springer-Verlag, 1996.
16. T. J. Mowbray and R. Zahavi. *The essential CORBA systems integration using distributed objects*, 1995.
17. H. J. Müller. Negotiation principles. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*, pages 211–230. Wiley Interscience, 1996.
18. H. Raiffa. *The Art and Science of Negotiation*. Harvard University Press, 1982.
19. J. S. Rosenschein and G. Zlotkin. *Rules of encounter: Designing conventions for automated negotiation among computers*. MIT Press, 1994.
20. J. Searle. *Speech acts*. Cambridge University Press, 1969.
21. N. Singh, M. R. Genesereth, and M. Syed. A distributed and anonymous knowledge sharing approach to software interoperation. *International Journal of Cooperative Information Systems*, 4(4):339–367, 1995.
22. R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man and Cybernetics*, 11(1):61–70, 1981.
23. H. Takeda, K. Iino, and T. Nishida. Agent organization with multiple ontologies. *International Journal of Cooperative Information Systems*, 4(4):321–337, 1995.
24. J. M. Tenenbaum, J. C. Weber, and T. R. Gruber. Enterprise integration: Lessons from shade and pact. In C. J. Petrie, editor, *Enterprise Integration Modelling: Proceedings of the First International Conference*, pages 356–369. MIT Press, 1992.
25. D. Zeng and K. Sycara. How can an agent learn to negotiate? In J. P. Müller, M. J. Wooldridge, and N. R. Jennings, editors, *Intelligent Agents III — Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages (ATAL-96)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, Heidelberg, 1996. In this volume.

This article was processed using the LaTeX macro package with LLNCS style