Autonomous Agents for Business Process Management[1]

N. R. Jennings[1], T. J. Norman[1], P. Faratin[1], P. O'Brien[2] and B. Odgers[2]

[1] Dept. Electronic Engineering, Queen Mary & Westfield College, University of London, London E1 4NS, UK.
{N.R.Jennings, P.Faratin, T.J.Norman}@qmw.ac.uk

[2] BT Research Labs, Martlesham Heath, Ipswich, Suffolk IP5 7RE, UK.
{paul, briano}@info.bt.co.uk

Complete mailing address:

Professor Nick Jennings
Department of Electronic Engineering
Queen Mary & Westfield College
University of London
London E1 4NS
UK
tel.: +44-171-975-5349
fax: +44-181-0259

[1]. ADEPT was a collaborative project under the DTI/EPSRC Intelligent Systems Integration Programme (ISIP). The project partners were BT Laboratories, ICI Engineering, Loughborough University, and Queen Mary and Westfield College.

Abstract: Traditional approaches to managing business processes are often inadequate for large-scale, organisation-wide, dynamic settings. However since Internet and Intranet technologies have become widespread, an increasing number of business processes exhibit these properties. Therefore a new approach is needed. To this end, we describe the motivation, conceptualisation, design and implementation of a novel agent-based business process management system. The key advance of our system is that responsibility for enacting various components of the business process is delegated to a number of autonomous problem solving agents. To enact their role, these agents typically interact and negotiate with other agents in order to coordinate their actions and to buy in the services they require. This approach leads to a system that is significantly more agile and robust than its traditional counterparts. To help demonstrate these benefits, a companion paper describes the application of our system to a real-world problem faced by British Telecom.

1 INTRODUCTION

Successful companies organise and run their business activities in an efficient manner. Core activities are completed on time and within the specified resource constraints. However to stay competitive in today's markets, companies need to continually improve their efficiency—business activities need to be completed more quickly, to higher quality and at lower cost. To this end, there is an increasing awareness of the benefits and potential competitive advantage that well-designed business process management systems can provide. Such systems can substantially improve efficiency by ensuring that business activities are better scheduled, executed, monitored, and coordinated.

The design and implementation of corporate-wide business management systems is a complex activity. The software has to support the distributed design and operation of many concurrent activities that are highly interdependent. Moreover, many of the activities have a real-time component, require the ability to access legacy software, and need context-dependent execution (i.e. their operation depends on the state of previous activities and of the environment—they are reactive systems [37]). In short, business management is a demanding domain that requires state-of-the-art software solutions. In this work, it was decided to conceptualise, design, and implement the business process management system using an *agent-based* approach. Thus in project ADEPT (Advanced Decision Environment for Process Tasks) the business process is viewed as a collection of autonomous problem solving entities that negotiate with one another and come to mutually acceptable agreements that coordinate their interdependent sub-activities. The main advantages of this approach over more traditional counterparts such as management information systems, workflow management, and enterprise integration are that it offers greater flexibility, agility, and adaptability.

The contribution of this work is in two main areas—business process management systems and agent-based systems. In the former case, this work represents a novel means of conceptualising and implementing software solutions. The insights gained in this work will assist designers of business process management systems in evaluating the appropriateness and benefits of the agent paradigm, in identifying the potential pitfalls, and in offering guidance on how to structure their applications. In the latter case, the work represents one of the few applications of multi-agent techniques to real-world problems (this has been identified as a major shortcoming of the discipline to date [16]). This work also makes contributions to the field of automated negotiation; previous work either made assumptions that are unrealistic for practical implementations or failed to adequately capture the richness of negotiation required in practical applications.

The remainder of the paper is structured as follows. Section 2 describes the domain of business process management, outlines the rationale for an agent-based solution and identifies the solution's key abstraction mechanisms.

Section 3 details ADEPT's system structure and agent architecture. Section 4 places this work in context by describing other approaches to business process management and related work in agent systems. Finally, section 5 provides some recommendations as to the use of agents in business process management and highlights open issues that need to be more fully addressed. A companion paper then demonstrates how the concepts described herein have been applied to a British Telecom (BT) business process for providing a quote for installing a network at a customer's premises.

2 AGENT-BASED BUSINESS PROCESS MANAGEMENT

This section introduces the domain of business process management (section 2.1), presents the case for an agent-based solution (section 2.2), identifies and justifies ADEPT's key conceptual components (section 2.3), and indicates how these components are used to build agent-based solutions (section 2.4).

2.1 The Basics of Business Process Management

Medina-Mora *et al.* [28] categorise processes in an organisation into material processes (the assembly of physical components or the delivery of physical products), information processes (related to the automated and partially automated tasks that create, process, manage, and provide information) and business processes (market-centred descriptions of an organisation's activities, implemented as information processes and/or material processes). In more detail, a business process can be split into a number of constituent components (figure 1).

<INSERT FIGURE 1 HERE>

Firstly, there needs to be a definition of the business process (left branch of figure 1). It describes, in some specification language, the activities that need to be performed, the participants who could or should perform them, and the interdependencies that exist between them[2]. Specification languages vary greatly in their details, but at a conceptual level they are broadly similar—they must provide a set of concepts useful for describing processes, their tasks, the dependencies between the tasks, and the required roles that can perform the specified tasks [8]. The activities in the process description may be automated, or involve humans interacting with computers.

Secondly, the business process needs to be executed and managed (right branch of figure 1). A software system needs to be devised that is capable of ensuring the process description is realised in practice. This system must: allow the human and the manual activities to be assigned appropriately; provide access to the software tools (e.g. databases, spreadsheets, design software, etc.) required to complete the tasks; and ensure the dependencies between the tasks are satisfied. Moreover, the software should transparently support multiple invocations (instances) of a given process and a given task.

The types of business process for which management systems have been devised vary enormously—from *ad hoc* interactions with few set paths and few set patterns of interactions, to repetitive, predictable processes with simple hard-wired coordination rules; from dealing predominantly with human-oriented activities (i.e. groupware appli-

[2]. An important aspect of successful business process management is the optimisation of the process. The first step of this endeavour is to understand the process as it currently operates (this is a typical systems analysis activity that involves interviewing people with expert knowledge about the process and studying relevant system documentation). The second step is to explicitly reconsider and redesign the process. Such business process re-engineering is typically carried out in order to increase customer satisfaction, improve the efficiency of business process operations, increase the quality of products, reduce costs, and/or meet new business challenges and opportunities by changing existing services or introducing new ones. The final step is to encode the revised business process description in the business process specification language. ADEPT deals exclusively with the final step and assumes that any necessary process re-engineering has already taken place.

cations) to entirely automated activities. See [8] for a detailed survey and classification. Here we are interested in large-scale (hundreds of activities), organisation-wide business processes. Our analysis identified the following key characteristics of this class of application [18]:

- The processes are dynamic and unpredictable. It is impossible to give a complete *a priori* specification of all the activities that need to be performed and how they should be ordered. Any detailed time plans that are produced are often disrupted by unavoidable delays or unanticipated events (e.g. people are ill or tasks take longer than expected). Coordination between the tasks also needs to be handled in a similarly flexible manner.

- The processes involve a mixture of human activities and automated tasks. The exact ratio varies between applications. Issues relating to the roles and interactions of humans in our system are dealt with in the companion paper.

- Multiple organisations may be involved in the process. Each organisation attempts to maximise its own profit within the overall activity.

- Processes are physically distributed; this distribution may be across one site, across a country, or even across continents.

- Within organisations, there is a decentralised ownership of the tasks, information and resources involved in the business process.

- Different groups within organisations are relatively autonomous—they control how their resources are consumed, by whom, at what cost, and in what time frame. They also have their own information systems, with their own idiosyncratic representations, for managing their resources.

- There is a high degree of natural concurrency—many interrelated tasks are running at any given point of the business process.

- There is a requirement to monitor and manage the overall business process. Although the control and resources of the constituent subparts are decentralised, there is a need to place constraints on the entire process (e.g. total time, total budget, etc.).

2.2 The Case for an Agent-Based Solution

The traditional (workflow) approach to business process management involves describing the entire process from a centralised perspective. That is, a complete list of all the activities and all the paths are provided, the criteria for following a particular path are specified, and the ordering constraints on the actions are given. Given this complete specification, the business process management system has the comparatively straightforward task of executing it. This approach works well for simple business processes. It has led to a range of commercial products (such as In-Concert [26], Staffware [50], and Action Workflow [29]) and has an estimated market size of $2.5 billion [27]. However for the class of business processes we considered, the following inherent shortcomings mean a traditional approach is unsuitable—it lacks [45]: (i) reactivity—workflow management systems require an *a priori* representation of the business process and all potential deviations from that process; (ii) semantics—many workflow management systems lack an appreciation of the content of a business process and do not make decisions based on the nature of the information that is generated; (iii) extensibility—many systems are not extensible on line; (iv) resource management—workflow management systems do not control the resourcing of a business process and so they rely on the process being fully dimensioned beforehand; and (v) heterogeneity—workflow management systems tend to take a centralised view with a single management engine that does not operate across multiple-server platforms or multiple-client operating systems.

To overcome these limitations a fundamentally new approach is needed. Our approach is to devolve responsibility for enacting specific business process activities to the constituent components, rather than maintain it centrally, and to make these components more active. Thus each of the business processes' main activities are assigned to a particular problem solving entity, and that entity is responsible for ensuring the activity is fulfilled within the specified constraints. The means by which the activities are performed are left to the responsible entity to determine. In many cases, a responsible entity needs the services of others to achieve specific sub-activities and these interactions may again involve devolving responsibility. Thus, delegation can continue through many levels of nesting.

Devolved responsibility means that the business process management system needs to be considerably more sophisticated than its traditional counterpart. Many decisions that are traditionally made in the process description at design-time are now moved to the execution system and determined at run time. Thus, instead of executing a fully defined process description, the execution system has to determine which activities should be performed, how much resource each activity can consume, who should perform them, when they should be performed, and how any interdependencies should be resolved.

Given this conceptualisation, a natural way to design and implement the business process management system is to make each responsible entity an autonomous agent. Such agents have specific goals to achieve and interact with one another to manage their interdependencies. In this context, an agent can be viewed as an encapsulated problem solving entity that exhibits the following properties [49]:

- *Autonomy*: agents perform the majority of their problem solving tasks without the direct intervention of humans or other agents, and they have control over their own actions and their own internal state.
- *Social ability*: agents interact, when they deem appropriate, with other agents in order to complete their problem solving and to help others with their activities.
- *Proactiveness*: agents take the initiative and exploit unexpected opportunities where appropriate.
- *Responsiveness*: agents perceive their environment and respond in a timely fashion to changes in it.

Secondary factors that point towards agents as a suitable solution technology include: (i) the domain involves an inherent distribution of data, problem solving capabilities, and responsibilities (conforms to the basic model of distributed, encapsulated, problem solving components); (ii) the integrity of the existing organisational structure and the autonomy of its subparts needs to be maintained (appeals to the autonomous nature of the agents); (iii) interactions are fairly sophisticated, including negotiation, information sharing, and coordination (requires the complex social skills of agents); (iv) the problem solution cannot be prescribed entirely from start to finish (the problem solvers need to be responsive to changes in the environment and to unpredictability in the process and proactively take opportunities when they arise); and (v) the domain includes a number of legacy systems, especially databases, that need to be incorporated into the business process (wrap up the existing code as an autonomous agent so that it can interact flexibly, through the agent's social abilities, with a range of new applications [7, 21]).

The main benefits of an agent-based approach over the traditional workflow view are as follows: (i) it offers greater flexibility since actions can be based upon the agent's current situation, rather than being prescribed in advance; (ii) it offers greater agility since new services can be added and configured with minimal effect on other agents; and (iii) it offers greater adaptability since an agent's choices can be guided by feedback received from previous invocations of particular paths through the business process. The relative drawbacks of an agent-based approach are that it offers a more fragmented view of the process and it is more difficult to ensure process-wide constraints are satisfied. However, on balance, it is felt that the benefits outweigh the drawbacks (this issue is returned to in section 5).

2.3 The Conceptual Framework

The main components of ADEPT's conceptual framework are interacting, autonomous agents that are responsible for performing particular activities (figure 2). Here we use the term "*service*" to denote activities (manual or automated) that an agent can manage. A service corresponds to a conceptual unit of problem solving activity in the business process. Examples of services include designing an artefact, providing an insurance quote for a customer, or reviewing a paper for a scientific journal. Services can be characterised as functions that take some (possibly no) inputs, undertake some computation (varying from a simple database look-up to the design of a chemical factory), and produce some (possibly no) outputs. Services go through three phases: (i) *specification*—detail what needs to be done (section 2.4); (ii) *provisioning*—determine which agent is responsible for executing the service and under what terms and conditions (section 3.3); and (iii) *management*—execute the service in line with the agreed terms and conditions (section 3.4).

<INSERT FIGURE 2 HERE>

The simplest service is called a *task* and it represents an atomic unit of problem solving in the ADEPT system. It may be performed by a human or by an automated program. These atomic units can be combined to form *complex services* by adding ordering constraints and conditional control. A *service description language* (SDL) has been developed to specify services (see section 2.4 for more details) and this language corresponds to ADEPT's process definition language (section 2.1).

As the agents are autonomous, there are no control dependencies between them. Therefore, if an agent requires a service that is managed by another agent it cannot simply instruct it to start the service[3]. Rather, service provisioning requires the agents to come to a mutually acceptable agreement about the terms and conditions under which the desired service will be performed (here such contracts are called *service level agreements* (SLAs)). The mechanism for making SLAs is inter-agent *negotiation*—a process in which parties verbalise contradictory demands and then move towards agreement by concession making or searching for new alternatives [32]. The basic context and form of all ADEPT's negotiations is identical. It involves connecting an agent that requires a service (the client) with one that is willing to provide it (the server). We term such negotiation *service-oriented* [43]. To perform such negotiations, agents need a *protocol* that specifies the role of the current message interchange—e.g. whether the agent is making a proposal or responding with a counterproposal, or whether it is accepting or rejecting a proposal. Additionally, agents need a means of describing and referring to the domain terms involved in the negotiation—for example, both agents need to be sure they are describing the same service even though they may both have a different (local) name for it and represent it in a different manner. This heterogeneity is inherent in most organisa-

[3]. This is one of the major features that distinguishes agent systems from object-oriented systems and more traditional forms of distributed computing [48].

tions because each department typically models its own information and resources in its own way (section 2.1). Thus when agents interact, a number of semantic mappings and transformations may need to be performed to create a mutually comprehensible *information sharing language* (see section 3.2 for more details).

In many business process applications there is a need to reflect the company's organisational structure when modelling the process and describing the behaviour of the problem solving components (section 2.1). Within the range of applications we considered, two types of relationship were observed: peer-to-peer and organisational hierarchies. To reflect these relationship types, ADEPT uses the notion of *agency* [36]. An agency is recursively defined: consisting of a single responsible agent, a, possibly empty, set of tasks that the responsible agent can execute, and a, possibly empty, set of sub-agencies (figure 3)[4]. For example, agency D has a single responsible agent that has two distinct tasks (TD1 and TD2) and three sub-agencies (E, F and G). The responsible agent represents the interests of the agency to its peers[5]. 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) typically behaves in a cooperative manner towards its responsible agent since this agent represents 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 [35]. This means when a responsible agent requests a service from one of its sub-agencies, the request is not refused without good reason. However, a sub-agency is not a subroutine. It retains a degree of local autonomy. For example, the manager of a design department may request a design engineer to work on a particular project. If the engineer can perform the task, the request will be accepted, but the conditions under which the request will be met are open to negotiation. In contrast, the relationship between peer agents is more open; an agent is not obliged to accept a request from a peer. An agent will come to an agreement with a peer agent if it is in its best interests to do so.

<INSERT FIGURE 3 HERE>

The agency structure also provides a mechanism for the encapsulation and abstraction of services. As an example, consider the agency illustrated in figure 3. Suppose that this diagram represents the structure of an organisation in which the design department is agency D. In this case, the responsible agent represents the department manager (i.e. the agent through which other departments (agencies A, B and C) may contact the design department), sub-agency E represents a single design engineer that is capable of performing two distinct tasks (or atomic services), and one of the other agencies (say F) represents a team of surveyors. Furthermore, suppose that the department manager has registered a "cost and design network" service that 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 the "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 the design that is proposed is consistent with the geographical requirements of the proposed network site. Therefore, for the manager to register the "cost and design network" service, at least one design engineer must register with its peers and responsible agent 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[6]. Note that it is neither necessary for the agent requiring the "cost and design network" service to know how this is achieved, nor is it nec-

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

5. Peer agencies are those with responsible agents that may communicate without crossing an agency boundary. For example, in figure 3 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.

6. 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.

essary 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.

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 unavailable).

2.4 Building Business Process Applications using ADEPT

The ADEPT system relieves some of the engineering burden of building business process applications by automating the allocation, scheduling and execution decisions (the management level of figure 4) that are specified at design time in traditional systems. (Refer to section 3 for details of *how* these activities are realised.) This inbuilt functionality enables the design engineer to concentrate on the specification of the application layer (figure 4) of an ADEPT implementation[7]. This process involves a number of constituent activities. Firstly, specifying services using ADEPT's SDL and determining their distribution between the system's agents and agencies. Secondly, defining the SLA template that represents, on a per service basis, the issues that need to be settled during a particular service-oriented negotiation. Thus, for example, the SLA template for service S1 may specify that the names of the client and server, the name of the service, the price of the service, and the time at which the service is to be provided are required as slots in S1's SLA. Whereas for a different service, S2, the SLA template may additionally specify that service quality and service volume are also relevant negotiation issues. Thirdly, providing acceptability ranges for the SLA slots that are to be determined by inter-agent negotiation; e.g. the maximum and minimum price that can be paid for a particular service, the shortest time in which a service can be completed, and the maximum number of concurrent invocations of a given service. These *reservation values* represent the agent's domain knowledge of a service, and they are used to constrain the process of service provisioning. Finally, specifying the information models that agents use during information sharing so they are able to interoperate despite the heterogeneity that is present in their local representations.

<INSERT FIGURE 4 HERE>

Defining the SLA template and providing the reservation values for each of the services are fairly simple knowledge acquisition tasks in most cases (see the companion paper for more details). However using the SDL to specify how services are realised and how different information models are related are considerably more complex and time consuming activities. The latter issue is dealt with at length in section 3.2 and the former is discussed in the remainder of this section.

For each service an agent provides, an SDL description must be produced. This description consists of a name (unique for that service), a set of inputs, a set of outputs, a guard, and a body. The inputs specify the information used by the service. Inputs can be either mandatory or optional. A mandatory input must be provided for every

[7]. The management and application layers are supported by an agent infrastructure which provides basic interoperation capabilities between the heterogeneous and distributed components of the business process management system. In the current implementation, this infrastructure is based on DAIS [2], a CORBA-compliant (Common Object Request Broker Architecture) distribution platform [31]. However, the ADEPT system is not restricted to a CORBA platform. Any distribution platform may be employed, provided that mappings are available between it and the convergence layer (a layer that provides a technology neutral infrastructure interface).

invocation of the service. An optional input provides more information to the service provider; it may enable the service to be performed more quickly or to a better quality, but is not necessary. Inputs are also categorised by their origin; they may be provided by the client, the server or either. There are five inputs to the example service `Prepare_Table` (figure 5), four mandatory (`man`) inputs (one to be provided by the client (`cli`), one provided by the server (`ser`), and two that can be provided by either (`any`)), and one optional (`opt`) input that can be provided by either the client or the server. There are no such distinctions for the outputs of a service, they are all assigned a value by the server; e.g. the output information object `Home_Seat_Allocation`. The inputs and outputs are defined in terms of the information model of the agent that is responsible for the service (see section 3.2). By convention, the name of an information object is prepended by the name of the information model in which it is defined. For instance, the inputs and outputs to the `Prepare_Table` service are information objects defined within the server's `Home` information model.

<INSERT FIGURE 5 HERE>

A guard is a boolean condition relating to the state of the world in which the service can be executed. It is evaluated when the service is invoked. If it evaluates to false, the service fails without the body of the service being processed. For example, the service `Prepare_Table` requires that the number of guests is less than or equal to the number people that can be accommodated. If the guard evaluates to true, the agent starts executing the service's body.

The body of a service description specifies how the service is to be executed, and consists of the restrictions on the order of its component services (tasks being atomic services), the conditions under which the service will be deemed successful, and how information flows between those component services. The body is composed of a single block that may be composed of further nested sub-blocks. Each block has the following syntax[8].

```
<block-type> `:' <block-identifier> `{' <execution-list> `} ->'
     <completion-expression>
```

The `<block-type>` is one of `sequence` (sequence of services), `can-para` (services can be performed in parallel), `must-para` (services must be performed in parallel), and `loop` (service iterates until some condition holds). Figure 6 consists of a `sequence` block, and `can-para` and `must-para` sub-blocks. The `<block-identifier>` can be used outside the block to refer to the completion state of that block. For example, the block identifier `prepare` identifies a `can-para` sub-block. This block identifier is used in the completion condition of the `meal` block, and will have the value `true` if the block to which it refers has been successfully completed, `false` if it has failed and `unknown` if it has not yet been commenced, or is in the process of being executed.

<INSERT FIGURE 6 HERE>

The `<execution-list>` is a comma separated list of services, conditionals and blocks. In this example, the execution list of the block `meal` is a sequence of a `must-para` sub-block `organise`, a can-para sub-block `prepare`, a single service `Eat_Meal`, and another `can-para` sub-block `clean_up`. The execution list may also include conditional statements. These statements can be used to test a piece of information. For example, the

8. A procedural language is not used because such languages typically require a rigorously specified flow of control. Since the body is executed by an autonomous agent in an unpredictable environment, it is felt that such control decisions are best left to the agent to determine at runtime (rather than being dictated by the designer at compile time). Thus, in ADEPT's SDL, the body specifies a partial flow of control with some restrictions on the order and the degree of concurrency of the execution and the completion expression supplies the agent with the completion logic of the block (in terms of `success`, `fail`, and `unknown`). It is then up to the agent to complete the service by the most appropriate means given its current circumstances.

sub-block `organise` tests whether there are any friends to ask for a meal. If the set of friends is non-empty, the identifier `have_friends` evaluates to `true`, otherwise it evaluates to `false`.

Every block, service, and conditional has a *completion state* that can evaluate to one of `success`, `fail`, or `unknown`. The completion state expresses whether the block, service or conditional completed successfully or not. If a block has not yet been executed, or if it is still being executed, its completion state is `unknown`. A loop block type may also return `unknown` when it has finished executing; if `unknown` is returned, the block is executed again. A conditional may return `unknown` if it relies on some information object that does not exist, or on an attribute of an information object that has an unknown value. As services get executed and information values change, the completion states change. The completion expression of the block `meal` states that the block is successfully completed if the meal is organised, prepared, eaten and cleaned up. If one of these components fails, then the whole block fails.

Services are called by referring to them by name and providing sufficient parameters for them to be executed. For example, the service `Prepare_Table` is called with a single parameter. The parameter is the `choice` output from the `Plan_GuestList` service, specified using the syntax `Plan_GuestList::choice`. (Note that the keyword `service` refers to the service in which this block is situated, and hence the information `service::friends` is the friends input to this service.)

Once the application layer has been instantiated, the business process is defined and can now be executed and managed through the functions of the ADEPT agents (which are described in the following section).

3 REALISING THE AGENT FUNCTIONALITY

This section describes how ADEPT's conceptual framework is realised. Section 3.1 describes the functional architecture of an ADEPT agent and section 3.2 deals with inter-agent communication issues. A description of how the architecture and the communication infrastructure support the agent's key activities of service provisioning (section 3.3) and service management (section 3.4) is then undertaken.

3.1 The Functional Architecture

ADEPT agents consist of a number of distinct functional modules[9] (figure 7) that are responsible for handling inter-agent negotiation (the interaction management module or IMM), for assessing the agent's current problem solving situation (the situation assessment module or SAM), and for executing services (the service execution module or SEM). These modules utilise persistent information about other agents in the environment that is stored in the acquaintance models (AMs) and information about themselves which is stored in the self model (SM). An agent sends and receives communications through its communication module or CM.

<INSERT FIGURE 7 HERE>

3.1.1 The self and acquaintance models

The self and acquaintance models are, respectively, an agent's repositories for knowledge about itself and others in its environment. In the self model, an agent maintains information such as the services that it can provide (and their reservation values), the resources available to it, and its current schedule of activity. In its acquaintance models, it stores information about the existence, and known capabilities of other agents, histories of past encounters with them and knowledge of how they model information.

9. This agent architecture is based on those of GRATE* [17] and ARCHON [20].

3.1.2 The interaction management module

The IMM is responsible for provisioning services through negotiation. Thus it both tries to procure the services the agent requires from its acquaintances, and decides which services the agent will provide to others and under what terms and conditions. In either case, services can be provisioned in two different modes depending on the client agent's intended pattern of usage and the server agent's scheduling capabilities: (i) *one-off*: the service is provisioned each and every time it is needed and the agreement covers precisely one invocation; (ii) *on-demand*: the service can be invoked by the client on an as-needed basis within a given time frame (subject to some maximum volume measurement).

The process of obtaining a service from another agent is initiated by the SAM. The SAM also indicates the desired mode of provision. The IMM is then responsible for determining which agent or agents to approach, which negotiation strategies to employ, etc. The IMM's decision making is supported by four types of information: scheduling constraints emanating from the SAM; knowledge an agent has about its preferences for particular agents, service prices, etc. (represented in its SM); the reservation values for each issue under negotiation (represented in its SM); and its knowledge of the capabilities of other agents (represented in its AM). With this knowledge and the agent's negotiation model (section 3.3), the IMM generates initial proposals, evaluates incoming counter-proposals, and produces counter-proposals of its own, all with the intention of reaching a mutually acceptable agreement for the provision of the required service.

The process of deciding which services to supply to others is initiated by the receipt of a proposal from another agent. This proposal is evaluated, in terms of whether it is feasible (defined by the SAM) and beneficial, and a decision is made as to whether it should be accepted, rejected, or modified. Again this process is expanded upon in section 3.3.

3.1.3 The situation assessment module

The SAM is responsible for assessing and monitoring the agent's ability to meet the SLAs it has already agreed and for assessing the agent's ability to meet any SLAs that are currently under negotiation. This involves two main activities: (i) the scheduling of services and tasks and (ii) the handling of high-level exceptions that occur when services and tasks are executed.

The scheduler maintains a record of the problem solving resources that the agent controls (i.e. the tasks that are available to it and the negotiated SLAs for which it is a client) and an indication of when each of these resources has been committed. This resource information is coupled with a coarse grain (approximate) scheduling algorithm to determine whether proposed SLAs can be satisfied in the service provisioning phase, and with a fine grain scheduling algorithm to determine which services should be executed at what times in the service management phase. Coarse grain scheduling is initiated when the server IMM receives a proposal (or counterproposal) from another agent and its aim is to decide whether the request is feasible. To ascertain this, the IMM asks its SAM whether the proposed schedule is likely to be acceptable. Clearly in advance of the situation, the SAM can only provide an estimate since the agent's circumstances may change between the point at which this check is made and the time at which the service will be required for execution. The SAM uses its knowledge of its current resource commitments and of the commitments that may follow from the agent's ongoing negotiations to generate one of the following responses: accept—it is likely that the proposed schedule will be satisfiable; reject—the agent cannot satisfy the proposed schedule; or revise—it is unlikely that the proposed schedule will be satisfiable, but the service is predicted to be satisfiable at the specified alternative. Fine grain scheduling is used when an agreed SLA is in place. It relates to the fixing of a particular time at which the service should be executed and a specific set of the agent's resources that will be deployed in this execution. Resources are therefore reserved before an agreed SLA is executed by the client. Furthermore, if the service is provisioned in an on-demand manner, the SAM uses the agreed volume of invocations to predict the future demand for its resources under that agreement.

The high-level exception handler analyses service execution exceptions as they occur (or even before in certain circumstances) and tries to formulate a set of recovery actions that will prevent the service from failing. For example, during the execution of a particular service, the SEM may realise that it requires a subsidiary service for which no SLA has yet been agreed (perhaps because this subsidiary service is on a little used path through the business process). The SAM then requests the IMM to arrange for this service to be made available if it is not already doing so. As a second example, if a service is delayed, then the SAM may decide to locally reschedule it (if this can be achieved without violating the existing SLA), to request that the IMM renegotiate the SLA (i.e. attempt to agree a new schedule with the service's consumer if the existing SLA cannot be met), or to terminate it altogether and pay any penalties specified in the SLA (if re-negotiation fails).

3.1.4 The service execution module

The SEM is responsible for managing services throughout their execution. This involves three main activities. Firstly, service execution management that involves parsing the service's SDL and firing off its constituent subparts according to the logic specified in the completion condition and the schedule specified by the SAM. In particular, this requires invoking, suspending, resuming and terminating tasks and services (this management is enacted via the CM (see section 3.1.5)). Each task or service execution instance is assigned its own processing thread and hence multiple services may be executed by a single agent at any one time. The SEM's second main activity is information management. This involves the routing of information between tasks, services and other agents during execution as specified in the SDL (see section 2.4). Finally, the SEM performs low-level exception handling. This involves monitoring the execution of tasks and services for unexpected events and then reacting appropriately. In the event of task failure, for example, the SEM may recover by attempting to restart the task if the present schedule can still be met, or if this cannot be achieved it will refer the problem up to the SAM for re-provisioning or renegotiation (see section 3.1.3).

3.1.5 The communication module

The CM is responsible for packaging messages destined for other agents in the shared communication language and information model, and the receipt and interpretation of messages from other agents and from its tasks (see section 3.2 for more details). During task management (e.g. the activation, suspension, or resumption of a task), messages are routed between the SEM and the tasks managed by that agent. During service execution management (e.g. the initiation or termination of a service being provided by another agent under an existing SLA), messages are routed between the agent's SEM and the service provider/consumer agent. During negotiation, messages are routed between the agent's IMM and the agent being negotiated with. In addition to this, the CM checks the validity of incoming messages to ensure they are correct in the present context (e.g. if the agent receives a counter-proposal from an agent it is not negotiating with, an appropriate error message is generated), and translates the content of the message between the shared information model and the agent's local information model.

3.2 Inter-Agent Communication

Agents communicate via an agent communication language (ACL). ADEPT's ACL consists of messages containing: one of a limited number of primitive message types, the identity of the sender, recipient (both agent identifiers) and thread of communication, the service concerned, and the information model with reference to which the contents of the message should be understood. Altogether there are 13 message types (table 1): 10 of which are used during negotiation (i.e. used by the IMM), and 3 are used during service execution (i.e. used by the SEM).

<INSERT TABLE 1 HERE>

In addition to the requirements that agents must share a common message syntax and interpret different message types in a uniform manner, information that is shared by two or more agents must have a common semantic interpretation. This is a significant problem when interacting agents do not necessarily model information in a consistent way[10]. Suppose, for example, that a network design department agent within a telecommunications company interprets the location of a customer's site to mean its postal address. However, a team of surveyors may understand the symbol "location" to refer to the site's grid reference on a standard Ordinance Survey map. If the design department requires a survey of a customer's site, how are these agents with their different models of information to understand one another?

Our approach is to use a common information model, through which agents may share information. This information model is built on a number of basic information object classes (e.g. `Adept_Boolean`, `Adept_Float`, `Adept_Integer` and `Adept_Char`), where each class is prepended by the name of the information model, in this case "`Adept`". Each class within the ADEPT information model contains a number of named slots containing further ADEPT information objects. For example, an object of type `Adept_Time` may be specified as follows:

```
(class Adept_Time
     (Adept_Integer year) (Adept_String month)
     (Adept_Integer day) (Adept_Integer hour)
     (Adept_Integer minute) (Adept_Integer second) )
```

The information model specification is then parsed to create a representation of that model in the native language of the agent (e.g. C, CLIPS, Prolog, etc.). Using the ADEPT model, an application-specific common information model is built that may, for example, specify information object classes for a customer's details, that may include their name, address, contact number, etc. A concrete example of such information objects for the BT application is given in the companion paper. In the unlikely event that the business process management system is being developed from scratch, the agent designer could choose to use this model within its domain tasks. However, in many cases business process tasks involve legacy software (e.g. a database of old customer records), that were built using a different model of information. Furthermore, if agents representing the interests of different companies are to interact, a common information model must reflect the information sharing needs between these organisations since it is extremely unlikely that they will use identical models internally. For these reasons, agents must have the ability to manage heterogeneous information models, and transform information that is expressed in task-specific or organisation-specific models to and from a common information model.

In transforming information between an internal and a common model, simple schema translations that specify the mapping between objects in each model may be used [20]. For example, the agent may be provided with a function

10. Here an information model should be understood as a specification of the symbols that an agent uses to make decisions.

that transforms its internal representation of a time point into an information object of the class `Adept_Time` illustrated above. Schema translations have the advantage of being computationally cheap, since they are basically a look-up table of mappings. However, they are costly to produce and maintain (schema translations must be specified for each agent within the system and significant modification may be required if either information model is changed) and are highly application dependant. Furthermore, if the required schema translation is not specified, the agent cannot communicate the information required. An alternative is to use deep representations of the internal and common information models (sometimes referred to as ontologies [9, 44]) to search for a transformation of an information object from one model to another. The initial production of ontology solutions is as costly as schema translation, but they have the advantage that they have the potential to be reused. However, reasoning with ontologies is computationally expensive. Given these design trade-offs, we investiagted a number of methods for the manipulation and combination of schema translations, and in the use of explicit ontological representations to generate novel schema translations. The method employed at present is to use an existing schema if one exists, and if not, attempt to build a schema by combining various components. For example, a schema for translating an interval represented by its start and end points may be constructed by using a schema for translating single time points. This is a area for further research.

3.3 Service Provisioning

The performance of the overall ADEPT system and of the various stakeholders that offer and consume services is intimately related to the efficiency of the inter-agent negotiation process. To prosper, agents need to be able to make beneficial agreements, in a reasonable time frame, without using excessive resources (either communication-related or computational). Moreover, this negotiation must be enacted in a decentralised fashion (i.e. without arbitration or third party intervention [38, 39]) since centralisation of processes creates bottlenecks and is susceptible to failure. Consequently, each agent has a negotiation capability; the agent's IMM.

To cope with the variety of negotiation situations in which an agent may find itself, the IMM needs a number of different negotiation strategies and tactics. These vary the agent's behaviour from competitive, through accommodative, to conciliatory [38]. Such a range of behaviour is necessary because negotiating with a peer differs from negotiating with a subsidiary agent, negotiating with an agent from an external organisation differs from negotiation with an agent from the same organisation, negotiation that requires a rapid agreement differs from negotiation in which time is plentiful, and so on. In more detail, a number of requirements for service-oriented negotiation emerged from the business process applications studied in ADEPT[11] [43]:

- A given service can be provided by more than one agent. The available services may be identical in their characteristics or they may vary along several dimensions (e.g. quality, price, availability, etc.).

- Individual agents can be both clients and servers for different services in different negotiation contexts.

- Negotiations can range over a number of quantitative (e.g. price, duration, and cost) and qualitative (e.g. type of reporting policy, and nature of the contract) issues. Each successful negotiation requires a range of such issues to be resolved to the satisfaction of both parties. Agents may be required to make trade-offs between issues (e.g. faster completion time for lower quality) in order to come to an agreement.

- The social context and inter-relationships of the participants influences the way agents negotiate. Some nego-

[11]. Although drawn from the business process domain, subsequent work in network management [5] has led us to believe that they are applicable for service-oriented negotiation in general.

tiations involve entities within the same organisation or within the same department and are generally cooperative in nature. Other negotiations are inter-organisational, and hence more competitive. Some groups of agents often negotiate with one another for the same service, whereas other negotiations are more infrequent.

- As agents are autonomous, the factors that influence their negotiation stance and behaviour are private and not available to their opponents (especially in inter-organisational settings). Thus agents do not know what utilities their opponents place on various outcomes, they do not know what reasoning models they employ, they do not know their opponent's constraints and they do not even know whether an agreement is possible at the outset (i.e. the participants may have non-intersecting ranges of acceptability).

- The communication channel between any two negotiating agents is private. Hence agents competing to provide the same services cannot check the behaviour of their opponents.

- Time is an important consideration in negotiation. Timings are important on two distinct levels: (i) the time it takes to reach an agreement must be reasonable; and (ii) the time by when the negotiated service must be executed is important in most cases and crucial in others. The former means that the agents should not become involved in unnecessarily complex and time consuming negotiations (the time spent negotiating should be reasonable with respect to the value of the service agreement). The latter means that the agents sometimes have hard deadlines by when agreements must be in place (this occurs mainly when multiple services need to be combined or closely coordinated).

In order to satisfy these requirements, a number of constituent components need to be designed and specified: (i) a protocol indicating when what messages can be sent during the negotiation (section 3.3.1); (ii) a structure representing the issues about which negotiation can take place (section 3.3.2); and (iii) a reasoning model to determine the agent's behaviour in its negotiations (section 3.3.3).

3.3.1 The Negotiation Protocol

All agents must adhere to ADEPT's negotiation protocol (figure 8) during service provisioning. The state transition arcs represent the participants' utterances: →<`primitive`> are those of the servers and <`primitive`>→ are those of the clients. Negotiation is initiated when a client utters `cando` (state 1 to state 2). The server can then either indicate that it is capable (state 2 to 3) or that it is not (state 2 to failure). If the server has acknowledged its capability or if the client knows it is capable because of information contained in its AM, the client may send out a proposal (state 3 to 4). The server can then either reject the proposal (state 4 to failure), accept the proposal (state 4 to 5) or counterpropose (state 4 to 6). If the server accepts, the client may either deny the contract to the server (state 5 to failure) or else confirm the contract (state 5 to success). Otherwise, if the server has counterproposed (state 4 to 6) then the client may either accept the new contract (state 6 to 7), reject it (state 6 to failure) or else counterpropose a new contract (state 6 to 4). There may be several transitions between states 4 and 6. If it is the client who eventually accepts the contract (state 6 to 7), then the server may decide to either award the contract to the client (state 7 to success) or else deny it to that client (state 7 to failure).

<INSERT FIGURE 8 HERE>

3.3.2 The Negotiation Issues

SLAs are the structures about which the agents negotiate (figure 9). They represent the bid on the table during negotiation and the final contract at the end of a successful negotiation. The SLA structure is derived from the types of legal contract that are often used to regulate current business transactions. The values contained in the slots represent the conditions for providing and consuming a service by a server and a client agent respectively. Agents can negotiate over multiple issues (values in different slots) at any one time.

<INSERT FIGURE 9 HERE>

In more detail, `service_name` is the service to which the agreement refers and `sla_id` is the SLA's unique identifier (covering the case where there are multiple agreements for the same service). `Server_agent` and `client_agent` represent the agents that are party to the agreement. `Delivery_type` identifies the way the service is to be provisioned (section 3.1.2). The SLA's scheduling information is used by the SAM and the SEM for service execution and management (see section 3.4)—`duration` represents the maximum time the server can take to finish the service, and `start_time` and `end_time` represent the time during which the agreement is valid. In this case, the agreement specifies that agent CHL can invoke agent NDD to cost and design a network whenever it is required between 09:00 and 18:00 and each service execution should take no more than 320 minutes. The agreement also contains meta-service information such as the volume of invocations permissible between the start and end times, the price paid per invocation, and the penalty the server incurs for every violation[12]. `Client_info` specifies the information the client must provide to the server at service invocation (in this case CHL must provide the customer profile) and `reporting_policy` specifies the information the server returns upon completion.

3.3.3 The Negotiation Reasoning Model

The reasoning model determines the agent's behaviour in a given negotiation context. It is responsible for: initiating negotiation to obtain a desired service; responding to proposals from other agents; determining when proposals should be accepted or rejected; and determining when counter-offers should be made and what these counter offers should be. All this reasoning is undertaken within the IMM (figure 10).

<INSERT FIGURE 10 HERE>

In more detail, the IMM has three reasoning components (see [43] for a formal specification) that are supported by information maintained in the agent models and the agent's working memory (section 3.3.3.1). The evaluation reasoner (section 3.3.3.2) takes proposals or counter-proposals coming in from other agents and determines whether they should be accepted, rejected or whether a counter-proposal should be generated. If a counter-proposal is appropriate, control is handed to the strategic and tactical reasoners to produce a response. The strategic reasoner decides, at a coarse level of granularity, how the agent should approach the particular negotiation (section 3.3.3.3). For example, whether it should be cooperative or competitive, whether time or resources is the primary consideration, etc. Finally, the tactical reasoner fills in the slots of the SLA in a way that enacts the chosen strategy (section 3.3.3.4).

3.3.3.1 Information Used in Negotiation

The reasoning components have two main repositories for information—the working memory and the agent models. The former represents transitory information related to ongoing negotiations, while the latter represents persistent storage of more stable information.

12. The legal enforcement and the actual payment of penalties is not handled by the ADEPT system at this time.

Information stored in the working memory is structured around the notion of a *negotiation thread*. A thread is essentially a record or history of utterances related to a particular negotiation need (i.e. finding a server for a particular service). It includes all the messages the agent has sent, all the messages the other agents have sent, which strategies and tactics the agent has deployed, the current status of all negotiation threads (in cases where the agent is managing multiple threads of negotiation for the same service), and the service's earliest start and latest end times.

In the context of supporting negotiation, the agent models represent the agent's (private) beliefs about itself and its environment. The AM includes: agency agents—unique names of individual members of the agent's agency; agency typology—the agent's relationships (peer, subsidiary agent, etc.) with other community members (see section 2.3); agency status—which agents are in the same organisation and which are external; agency capacity—which agents can provide services the agent needs; the negotiation protocol (figure 8); and interaction histories—persistent records of negotiation threads. The SM includes: service descriptions for the services the agent can perform itself, together with an indication of the number of concurrent invocations that are permissible; the commitments the agent has already made through its SLAs; the agent's reservation values for the services it consumes and provides; and the agent's preferences for the various issues under negotiation (expressed as a scoring function).

3.3.3.2 The Evaluation Reasoner

The evaluation reasoner becomes active when an agent receives a proposal or counter-proposal from another agent. Upon receipt of such a message, the agent computes the utility it attains for the proposal. It uses an additive scoring function [39] over each slot in the SLA, where each slot is assigned a weight representing the relative importance of that issue to that agent. For example, consider the case depicted in figure 10 where the NDD agent receives the SLA proposal a1001. NDD goes through each slot in the proposal and assigns a measure of desirability (a utility rating [15] between 0 and 1) to the value contained therein. The raw utility values are then multiplied by a weighting factor (that indicates their relative importance) and then summed over all the slots. This process produces a single utility value for the proposed SLA. In parallel, the agent sends the offer that has just been received to the tactical reasoner to see what offer the agent would produce next using its current strategies and tactics. Once computed, this offer is returned to the evaluation reasoner and rated using the aforementioned scoring function. If the utility of the offer the agent would have sent is less than or equal to the utility of the offer just received, the offer is accepted (provided it meets the scheduling constraints coming from SAM). Acceptance involves a conditional commitment by the server that it will execute the specified service under the SLA's terms and conditions. The commitment is conditional in that the client still has to confirm or deny the contract (figure 8). Assuming the client confirms the contract, it then terminates all other negotiation threads for the same service instance. The second outcome of the SLA's evaluation is that the proposal is rejected. This occurs either when the deadline for reaching an agreement has been reached or when another agent has been selected to perform the service. The final evaluation outcome is that the offer is neither accepted nor rejected. In this case, the agent generates a counter offer.

If a counter offer is to be made, the evaluation reasoner also makes an assessment of the opponent's negotiation behaviour in the current thread. Thus, evaluation is not only confined to the current offer instance, it also incorporates the relationship of that offer to previous ones in the thread. In particular, the agent classifies the behaviour of its opponent into one of three mutually exclusive states: i) `CONCEDING`—the utility to the recipient of the last offer is greater than the previous offer received from that agent; ii) `EXPLOITING`—the utility to the recipient of the last offer is less than the previous offer received from that agent; or iii) `STALEMATE`—the utility to the recipient of the last offer is the same as the previous offer received from that agent. As well as the direction of change, the agent uses the negotiation thread history to determine the rate of change of that state. Thus the agent calculates whether this conceding/exploiting is `INTENSIFYING`, `LESSENING`, or `CONSTANT`. These two pieces of information are then passed onto the strategic reasoner which uses them to determine whether its present strategy is being successful or whether a change is needed.

3.3.3.3 The Strategic Reasoner

The strategic reasoner is invoked by the evaluation reasoner in the case of an ongoing negotiation or by the SAM for new negotiations. In either case, the purpose of the reasoning at this level is to set broad guidelines about how the agent should behave in a particular negotiation context. In the current implementation, these guidelines relate to determining the relative importance of the three classes of behaviour that take time, resources, and an opponent's behaviour as the primary basis for computing an offer. Time is important when the negotiation has a deadline. Resources need to be considered so that the agent expends an amount appropriate to the value of the contract. The opponent's behaviour is considered to ensure the agent is not exploited during the negotiation. The relative importance of these three classes is expressed by assigning a series of weights to the alternatives.

For new negotiations, the agent receives information from the SAM about when the service is required (`HAVE-TIME`, `NOW`), uses AM information about the number of known suppliers of the service (`ONE`, `MANY`), and uses AM information about the agent's relationship with the potential service provider (`SAME-ORGANISATION`, `EXTERNAL-ORGANISATION`) to set the strategy.

The first strategic decision relates to the logistics of the negotiation: who to negotiate with; whether to negotiate with more than one agent; and if more than one agent is to be negotiated with, then should the negotiation proceed sequentially or in parallel. If there is only one service provider then the agent has no real choice to make at this level. However when there are several providers, the agent uses the following heuristics to manage this aspect of the process (figure 11).

<INSERT FIGURE 11 HERE>

Having decided upon the logistics, the agent must determine how it is to behave. Example rules for setting an agent's strategy, along with their justification, are shown in figure 12. In addition to setting the strategy, the agent records its expectation of how the negotiation should develop in terms of the speed at which it will converge and the likely response of the opponent. This information is then used to monitor the progress of the ongoing negotiation.

<INSERT FIGURE 12 HERE>

For ongoing negotiations, the role of the strategic reasoner is to determine whether the current strategy is being successful (in terms of the agent's predictions about its development and in terms of the utility the agent is obtaining from the deal) in fulfilling the agent's negotiation objectives. Such monitoring is needed because the world in which the agent is operating is subject to change (e.g. the agent may require the service sooner/later than it estimated or a new provider for the service may be discovered) and also because operating a fixed, unchanging strategy means the agent is more open to exploitation by its opponents (since its behaviour is easier to predict). Strategy modification is triggered by two types of event: (i) whether there is a change in the agent's internal state (e.g. whether the time by which an agreement should be in place is becoming critical); and (ii) how the opponent is behaving (e.g. `CONCEDING`, `EXPLOITING`, `STALEMATE`, `INTENSIFYING`, `LESSENING`, `CONSTANT`). Example rules illustrating such strategy monitoring and modification are given in figure 13[13].

<INSERT FIGURE 13 HERE>

[13]. Note whenever the weights are changed, the agent re-normalises their values.

3.3.3.4 The Tactical Reasoner

The role of the tactical reasoner is to enact the high-level behaviour set by the strategic reasoner. The output of this level is a SLA which has values in each of its slots. Thus a tactic is a function which acts in line with the set strategy, to set a value for each SLA slot. For quantitative slot parameters, tactics have to select a value in between the allowable minimum and maximum value for that issue. For qualitative values, the tactics have to choose from a discrete range of alternatives—a process achieved by mapping the qualitative values onto the quantitative scoring function [3].

The way in which tactics differ is in how they go about computing a slot value. There are three main ways of coming to a value (more details of the operation and variety of tactics can be found in [43] and in section 4.3 of the companion paper):

• *Time-dependent tactics*: This family of tactics base their behaviour on the time remaining until an agreement must be in place. At their negotiation deadline all these tactics put forward their reservation values. However the way in which they concede to reach these values differs. There are two broad patterns of concession: (i) *boulware* [39]: maintain the offer until the time is almost exhausted and then begin to concede up to the reservation value; and (ii) *conceder*: move rapidly to the reservation value.

• *Resource-dependent tactics*: This family of tactics base their behaviour on the amount of a given resource remaining. The property of these tactics is that they model the urgency of the deal as: i) the resources become scarcer, ii) the willingness of other parties in negotiation decreases (measured as an increase in the length of the negotiation thread) and iii) the computational load on the agent increases. The actual relationship is that the quantity of time left in negotiation is proportional to the number of agents in the negotiation and inversely proportional to the length of the negotiation thread. Thus, the more agents who are potentially available to perform the service, the longer the agent can afford to negotiate. But the longer the duration of the negotiation, the more urgent the need for an agreement becomes.

• *Behaviour-dependent tactics*: This family of tactics base their behaviour on how their opponent behaves during the ongoing negotiation thread [1]. The tactics within this family differ in *which* aspect of their opponent’s behaviour they imitate, and to *what degree*. There are three ways in which behaviour can be imitated: i) Relative Tit-For-Tat; ii) Absolute Tit-for-Tat; and iii) Averaged Tit-For-Tat, where other’s behaviour is, respectively, imitated proportionally, absolutely and in an averaged fashion.

Each of the families computes a value for each of the negotiation issues based upon their particular perspective (figure 14). The three values for each issue are then combined, according to the relative weightings set by the strategic reasoner, to provide a single value which is the one put forward for that issue.

<INSERT FIGURE 14 HERE>

3.4 Service Management

Having made an agreement to provide a service, an agent must then attempt to honour it. This process involves two principal activities: (i) scheduling the service (and its constituent sub-parts) in accordance with the terms and conditions of the SLA; and (ii) executing the service.

When an agent agrees to provide a service to an acquaintance under a specific SLA, the SAM performs fine grain scheduling (see section 3.1.3) to determine when each of the service's constituent components should be executed and which problem solving resources should be deployed[14]. The SAM uses two techniques to proactively schedule tasks and services before they are required: i) *sequential* scheduling and ii) *look-ahead* scheduling. The former is used in association with on-demand provisioning. Each on-demand task or sub-service associated with an agreed service is scheduled according to the service description before the service is executed for the first time. The latter is associated with the scheduling of component parts of one-off services. As a task or service is being executed the next component of the process is proactively scheduled for execution (i.e. the SDL is parsed one step ahead and the next step is scheduled while the previous one is being executed).

For both sequential and look-ahead scheduling, the SAM sets up reservations that are used by the SEM for the actual execution of a service or task. A reservation principally associates a particular task instance or service with a unique SLA. The SAM sets up reservations to provide sufficient resources to comply with this agreement, considering components such as the agreed volume of invocations and the permissible degree of concurrency. Each agent may have its own scheduling methodology and could be linked to a generally available or legacy scheduler. The reservations are processed in two ways according to whether the service is one-off or on-demand. In the former case, the SEM invokes the resource that has been set aside in the fine grain scheduling phase and on completion informs the SAM of the service's end result. The SAM then removes the reservation and frees up the resource. In the latter case, a number of reservations (based on the agreed volume of invocations) are initially set up and noted as earmarked. This implies that the resources can be used with priority given to the associated SLA. When the SEM uses one of these resources, the reservation is set to committed, thus barring any other process instance from using it. On completion, the reservation is reset to earmarked ready for re-use. Only when the associated on-demand SLA is completed is the SAM informed and the reservations removed and the resources freed.

Due to the dynamic nature of the domain, exceptions often arise during service execution. The SEM monitors the execution of tasks and detects when something has gone wrong. If the exception can be handled by the SEM (section 3.1.4) then it is. Otherwise the SEM informs the SAM of the problem. These exceptions can be of two types: (i) functional or (ii) resource related. A functional exception indicates that the particular activity being performed by a resource has gone into a state of error. In this case, the SAM can either attempt to re-schedule the same task instance or re-resource the task by scheduling another instance of the same task type. Both options are usually considered and the SAM decides which is likely to be the most effective in a given situation. For example, re-resourcing is quicker if there are spare resources immediately available, whereas re-scheduling does not require new resources to be considered. A resource related exception indicates that a failure has occurred with the underlying resource of a task. When this occurs, the task instance needs to be de-allocated and all the tasks that have been provisioned to that instance need to be re-resourced. When the failed task instance comes back on-line, the SEM informs the SAM that then re-allocates the task instances so making them available for subsequent scheduling activities. If the SAM cannot handle the exception within the SLA's agreed times, it instructs the IMM to see if it can re-negotiate the SLA. If a new agreement can be reached, then a revised version of the SLA is instantiated. If a new agreement cannot be reached then the service fails. In this case, the server has to pay any penalty specified in the SLA.

[14]. Within an agent the problem solving resources are dynamically determined. Thus, when an agent is initialised, it has no knowledge of its resources. When tasks are initiated, the SAM forms a resource list containing the instance names associated with a certain task type. Only then can specific task instances be provisioned. If a task instance fails, it is removed from the task resource list and not provisioned in any future agreements. If a task becomes available during the running of the business process, the SAM initiates a search for any incomplete agreements and, if possible, uses this resource. If not, it is added to the appropriate resource list in anticipation of future use

4 RELATED WORK

There are a number of research areas that impact upon, and are related to, the work described in this paper. Here we focus on the three that are most closely related to ADEPT's key components; namely: (i) extant workflow systems (section 4.1); (ii) automated negotiation by autonomous agents (section 4.2); and (iii) techniques for allowing agents with heterogeneous information models to interoperate (section 4.3). These areas are dealt with in turn.

4.1 Extant Workflow Systems

At the time of writing, there are no commercial systems that support business process management using a meaningful notion of agenthood. Existing workflow management systems offer limited support and minimal flexibility during process enactment. In situations where a business process is fully resourced (dimensioned) and every conceivable outcome can be considered and controlled, then conventional distributed computing techniques and traditional workflow systems are adequate. If, however, the system has to cope with undefined errors or failures, and there is a need for dynamic re-configuration of resources, then the ADEPT approach is more flexible and robust.

The core functionality of a traditional workflow system is to automate the execution of a sequence of tasks in support of a business process. Typically, workflow systems consist of an engine that executes business tasks in a predefined order (as specified in a script [13]). The ADEPT system subsumes this functionality in its SDL and in the SEM's execution of these service descriptions. However, unlike workflow management systems, ADEPT also performs both resource management and sophisticated exception handling:

- ADEPT agents have the ability to perform explicit resource management; they control and reason about the systems, databases, equipment and people that make up an organisation. Traditionally business process management systems do not provide an inbuilt capability for such direct resource management. Instead processes have to be resourced and dimensioned prior to enactment. The ADEPT approach means the system can be far more responsive to unexpected or unusual patterns of resource availability.

- Presently in workflow systems, exception handling is managed by explicitly representing an alternative path through the business process. In ADEPT, agents dynamically attempt to renegotiate and re-resource the process task in order to resolve exceptions. This approach allows agents to react in a context dependent manner to circumstances where the type of corrective action might vary depending upon the availability of resources and the task's criticality within the process.

The final differentiator is that workflow management systems tend to operate with a central workflow engine that monitors all the events in the system. This type of architecture is limiting when a business process spans a large enterprise. ADEPT takes a distributed, and hence more robust and scalable approach, where the disparate components of a business process are each represented by an agent. Agents can be distributed either logically or physically throughout an organisation.

Given the limitations of current generation workflow systems, a number of researchers have considered using multi-agent systems for various aspects of business process management. Hall and Shahmehri [11] use agent technology to enable both expert (e.g. a manager or business process engineer) and non-expert users involved in a business process to influence the design and modification of that process. A language is presented for the description of processes and tasks to enable automatic reasoning about the operation of the business process, and hence facilitate the reuse of existing processes. A task is represented by a role that indicates who should execute the task, a set of preconditions, a task description, and a set of stop conditions. This is similar to the structure of a service description in the ADEPT model: the mandatory inputs to the service are preconditions for the execution of the service, a description of the processes involved in executing that service is provided in the body, and the completion conditions have a similar role to stop conditions. Such a rigorous description of processes and tasks provides an agent-based business process management system with the potential to modify the business process (possibly with reference to an expert) in response to changing circumstances. Therefore the use of agent technology to enable modification of a business process is complimentary to agent-based business process management systems such as ADEPT.

A federation-type architecture [7, 44, 46] provides an alternative method for organising multi-agent systems for the management of business processes. Agents are organised into groups, each group being associated with a single "facilitator"[15] to which an agent surrenders a degree of autonomy. A facilitator serves to identify agents that join or leave the system and manages direct communication between participating agents; functions that are similar to those provided by the DAIS ORB [2]. In addition, the facilitator provides anonymous communication (i.e. agents are informed of events in which they have registered an interest 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. This architecture enables agents to communicate without concern for the particular syntactic 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 its facilitator. During negotiation, participating agents require secure communication, but this direct communication is enabled and managed by one or more facilitators. These facilitators represent the interests of many different agents. Therefore, a facilitator that is managing direct communication between negotiating agents must be trusted to act in the interests of these agents even if this conflicts with other interests it is representing. At present, the federation architecture has been used predominantly for the interoperation of purely cooperative agents at the team level of an organisation (e.g. SHADE and PACT [46]). Such security issues must be addressed if this architecture is to be employed in business process management where more than one organisation is involved. 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.

Mobile agents have also been proposed as an approach to the management of workflow in business processes. Merz et al. [30] argue that the use of this technology means that only those organisations that require services from others are required to implement mobile agents. Other organisations need only accept the arrival of mobile agents and handle their requests; i.e. an organisation may participate in either a passive or an active manner. Each mobile agent is an encapsulated, autonomous unit, and therefore can participate in functions such as negotiation without relying on a facilitator-type agent (an advantage over the federation architecture). A further advantage of mobile agents that is often claimed is that they reduce communication overhead, but this is yet to be shown in practice; a reasonably sophisticated mobile agent may take a considerable time to transmit over a network. One potentially serious problem with mobile agent technology in the management of business processes is the lack of security. To participate in a mobile agent based business management system, an organisation must allow sophisticated programs from another, possibly competing, organisation to execute on their local machines. Therefore, for mobile agents to be a good implementation choice in this case, these security issues must be addressed.

15. Takeda et al. [44] refer to a "facilitator", "mediator" and "ontology server" in their architecture. Together, these three units perform the same function as a facilitator in the federation architecture described by Genesereth and Ketchpel [7]

In this context, we are interested in designing and building well-engineered coordination techniques that increase the efficiency and flexibility of task allocation. To this end, our approach has been to adopt and, where necessary, adapt tools and techniques from game theory and from the social sciences.

The central aim of game theory is the specification of rational equilibrium behaviours (or strategies) when multiple agents interact [12, 40]. Thus, game theoretic models are an obvious source of inspiration for our work. These models are not only analytically useful, but they also have several desirable properties. For example, Rubenstein's model of Alternative Offers [41] takes the passage of time into consideration, respects our negotiation protocol, and it can be shown that in such games agents have a simple and stable negotiation strategy that results in efficient agreements without delays [23]. So why don't we use game theoretic techniques directly? To answer this question we first consider the basic assumptions that most game theoretic models make and then consider what would be involved in applying these techniques in the ADEPT context. Most game theory models rest on the assumptions that the negotiating agents are: self interested (utility maximisers), computationally and communicatively unbounded, and rational. Moreover, the agents are assumed to have a set of alternatives that are fixed and known to all agents and each agent's risk attitude and utility function are fixed and known to all the agents that are involved in the decision making. Thus in order to apply these models a designer must [24]: (i) choose a strategic bargaining model; (ii) map the application problem to the chosen model's nomenclature; (iii) identify equilibrium strategies; (iv) develop simple search techniques for appropriate strategies; and (v) provide utility functions. Whilst choosing a strategic bargaining model and mapping it to an application may not be too difficult for someone proficient in game theory techniques, it is not clear how to design and implement equilibrium strategies in ADEPT when there can be infinitely many possible agreements. (Recall game theory requires all the agreements to be known in advance before equilibrium strategies can be proven.) The aforementioned assumptions also mean that most game theoretic models do not consider the computational and communication complexities that are so important in practical applications. Furthermore, in our context, each agent's utility function, set of alternatives, and risk attitude are private information (especially in inter-organisational settings) and even if such information was publicly available it would soon become intractable for large games.

The ADEPT negotiation model has also been influenced by social science models of negotiation. These models do not make the restrictive assumptions of game theory and they attempt to identify and *describe* behaviours that may achieve satisfactory outcomes [6, 22, 38, 42] (rather than *prescribe* behaviour like the game theory models). For example, the competitive, accommodative and conciliatory negotiation behaviours in ADEPT are heuristics that agents use as search operators to prune their set of possible actions. However, although such models are inspired by successful human negotiation behaviour, they suffer from the fact that the system's behaviour cannot easily be predicted. Thus, considerable effort is required, through simulation and empirical evaluation, before a negotiation mechanism design leads to a stable and predictable system. In our case, this experimentation showed that the ADEPT negotiation model converged in the majority of circumstances and that the communication and computational overheads were acceptable for this application [3]. Furthermore, subsequent theoretical analysis has demonstrated the validity of these results for a subset of the ADEPT scenarios [47].

In summary, we have used elements of game theoretic negotiation (such as utility functions and rational choice) as the basis for the IMM's decision making. These functions have then been augmented by work emanating from the social models that has provided negotiation heuristics to guide the IMM in its process of setting up negotiation, generating offers and counter-offers, and in monitoring and modifying its strategy over time.

4.3 Information Interchange

In common with a number of related enterprise integration projects [7, 44, 46], ADEPT agents share information that is expressed in a common information model (often referred to as an ontology [9]). The development of ontologies for reuse is an important research area in distributed system development. It is generally accepted [14, 34] that a domain ontology should not be written from scratch; it should be a fusion of existing ontological information. A number of ontologies have been developed both within the domain of enterprise integration [4] and specifically related to activities such as planning [33]. These ontologies are intended for reuse in other systems [25]. This work, and tools such as the Stanford KSL Ontology Editor[16], should serve to reduce the time consumed in the development of an ontology for a specific application. With the reduction of this overhead, the use of explicit ontological representations of information within a system, as opposed to hand-coded schema translations, becomes more attractive.

Motivated by the advantages and disadvantages outlined in section 3.2, a number of methods for building schema translations were investigated, and a simple initial mechanism employed. The hybrid approach advocated here has the potential to benefit from the advantages of schema translations, and the formation of novel schema when required. However, further research into these techniques is required. Within the Carnot project [14], the Cyc [10] global ontology along with database schema are used as inputs to the semi-automatic, user driven Model Integration Software Tool (MIST), that produces articulation axioms (aka. schema translations). Articulation axioms map local schema to Cyc; the Cyc global ontology then functions as a shared information model. Within the more restricted domain of a business process, totally automating the formation of schema translations from ontological representations is more feasible. An agent that is able to communicate (and interpret) information for which it has no existing schema translation provides greater flexibility; e.g. the agent may be able to take advantage of new services (with slightly different information requirements) that are offered in the system. Thus, with the reuse of existing ontologies, and the use of these ontologies to automatically (or with some reference to a domain expert) generate schema translations, the disadvantages of using explicit ontological representations are reduced, and more flexible interaction between agents is possible.

5 CONCLUSIONS

This paper described the conceptualisation and implementation of an agent-based system for managing corporate-wide business processes. The ADEPT philosophy is founded upon two key notions: (i) devolving responsibility for provisioning and managing the business; and (ii) making the problem solving components reactive and proactive so they can respond to unexpected situations. To this end, this work can be viewed on three different levels, each of which represents increasing support for the realisation of business process management software systems:

(i) ADEPT as a *design* technology: ADEPT proposes a method of approach for structuring the design and development of business process management systems. It identifies the key concepts in this view as autonomous agents, negotiation, service provision, service level agreements, resource management, and information sharing. This view can be readily applied to other business process applications without being tied to the details of how they were realised in ADEPT.

[16]. The Stanford KSL Ontology Editor is a tool that supports distributed, collaborative editing, browsing and creation of ontologies represented in Ontolingua [9], a language designed for the representation of ontologies: http://www-ksl-svc.stanford.edu:5915/FRAME-EDITOR/&sid=ANONYMOUS&user-id=ALIEN

(ii) ADEPT as an *implementation* technology: As well as identifying a conceptual framework, the ADEPT system provides concomitant algorithms, interfaces, language definitions, and implementation structures. These definitions can be re-implemented in other programming environments to develop ADEPT-like agent systems for business process management.

(iii)ADEPT as a *solution* technology: The ADEPT programming environment can be re-used in other business management applications. In this case, the ADEPT design methodology is used to structure the application and the ADEPT software is used to implement it.

As we have indicated, devolving responsibility to autonomous agents offers many advantages over traditional workflow approaches. However there are two potential drawbacks of this approach (as mentioned in section 2.2). Firstly, it is more difficult to attain a coherent view of the entire business process—since its state is now distributed. To combat this, significant efforts were expended on a suite of visualisation tools that enabled the business process manager to view and re-construct the system's state from its constituent components (see [19] and the companion paper for more details). Secondly, given the autonomous nature of the problem solving components, there is a greater chance that the business process will fail to meet any overarching constraints placed upon its operation. This is because the business process is constructed through dynamic, on-the-fly agreements, rather than through preset routes. To minimise such difficulties, the negotiation strategies and tactics of the organisation's agents need to be carefully engineered so they maximise the chance of making agreements. Empirical work on analysing the properties and relative merits of different combinations of strategies and tactics is reported in [3] and theoretical work on a subset of these scenarios is reported in [47].

The two major technical advances achieved by ADEPT relate to the techniques developed for automated negotiation and the techniques for information sharing between agents with heterogeneous information models. In the former case, our approach allows agents to exhibit a range of negotiation behaviours depending upon the context in which they find themselves. In the latter case, a pragmatic, hybrid solution that combines the speed of schema mappings with the flexibility of working with ontologies was adopted.

As already indicated, ADEPT can be used as a solution technology for real world applications. In particular, it has been applied to a BT business process of providing a quote to install a customer's network. Details of this application are given in the companion paper. That paper serves two main purposes: (i) it illustrates the concepts described herein; and (ii) it offers insights into how the ADEPT approach can be applied in practical situations.

References

[1] R. Axelrod, *The Evolution of Cooperation*. Basic Books, 1984.

[2] DAIS: A Platform To Build On, ICL Corporate Systems Publications, Version 2.1, 1994.

[3] P. Faratin, C. Sierra, and N. R. Jennings, Negotiation Decision Functions for Autonomous Agents. *Int. Journal of Robotics and Autonomous Systems*, **24** (3-4) 159-182, 1998.

[4] M. S. Fox, and M. Gruninger, Ontologies for Enterprise Integration, *Proc. of the 2nd Conf. on Cooperative Information Systems*, Toronto Ont, 1994.

[5] FIPA97 Specification—Part 7, Network Management and Provisioning, Foundation for Intelligent Physical

Agents, 1997. `http://www.fipa.org`

[6] R. Fisher and W. Ury, *Getting to Yes: Negotiating Agreement without Giving in.* Houghton Mifflin, 1981.

[7] M. R. Genesereth and S. P. Ketchpel, Software agents. *Comms. of the ACM* **37** (7) 48-53, 1994.

[8] D. Georgakopoulos, M. Hornick, and A. Sheth, An Overview of Workflow Management. *Int Journal of Distributed and Parallel Databases* **3** 119-153, 1995.

[9] T. Gruber, A translation approach to portable ontology specifications. *Knowledge Acquisition* **5** 199-220, 1994.

[10] R. V. Guha and D. B. Lenat, CYC: A Mid Term Report. *AI Magazine* **11** (3) 32-59, 1990.

[11] T. Hall and N. Shahmehri, An intelligent multi-agent architecture for support of process reuse in a workflow management system, *Proc. of 1st Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology*, London, UK, 331-343, 1996.

[12] J. C. Harsanyi, *Rational Behaviour and Bargaining Equilibrium in Games and Social Situations*. Cambridge University Press, 1977.

[13] D. Hollingsworth, The Workflow Reference Model. The Workflow Management Coalition, 1994.

[14] M. N. Huhns, M. P. Singh, T. Ksiezyk and N. Jacobs, Global Information Management via Local Autonomous Agents. *Proc. 13th Int. DAI Workshop*, Seattle, WA, 153-174, 1994.

[15] R. Jeffrey, *The Logic of Decision.* University of Chicago Press, 1983.

[16] N. R. Jennings, *Cooperation in Industrial Multi-Agent Systems*. World Scientific Publishing, London, 1994.

[17] N. R. Jennings, Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence* **75** (2) 195-240, 1995.

[18] N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien and M. E. Wiegand, Agent-Based Business Process Management. *Int Journal of Cooperative Information Systems* **5** (2&3) 105-130, 1996.

[19] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, M. E. Wiegand, C. Voudouris, J. L. Alty, T. Miah, and E. H. Mamdani, ADEPT: Managing Business Processes using Intelligent Agents. *Proc. BCS Expert Systems 96 Conference* (Intelligent Systems Integration Programme Track), Cambridge, UK, 5-23, 1996.

[20] N. R. Jennings, E. H. Mamdani, I. Laresgoiti, J. Perez, and J. Corera, Using ARCHON to develop real-world DAI applications. *IEEE Expert* **11** (6) 64-70, 1996.

[21] N. R. Jennings, L. Z. Varga, R. P. Aarnts, J. Fuchs and P. Skarek, Transforming Stand-alone Expert Systems into a Community of Cooperating Agents. *Int. Journal of Engineering Applications of AI* **6** (4) 317-331, 1993.

[22] S. Kraus and D. Lehmann, Designing and building a negotiating automated agent. *Comput. Intell.* **11** 132-171, 1995.

[23] S. Kraus, J. Wilkenfeld and G. Zlotkin, Multi-agent negotiation under time constraints. *Artificial Intelligence* **75** 297-345, 1995.

[24] S. Kraus, Negotiation and cooperation in multi-agent environments. *Artificial Intelligence* **94** 79-97, 1997.

[25] F. Lehmann, CCAT: The Current Status of the Conceptual Catalogue (Ontology) Group, with Proposals. *Proc. 4th Int. Workshop on Peirce: A Conceptual Graph Workbench*, University of Maryland, 1994.

[26] D. McCarthy and S. Sarin, Workflow and Transactions in InConcert. Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society 16 (2) June 1993.

[27] S. McCready, There is more than one kind of workflow software. Computerworld, November 2, 1992.

[28] R. Medina-Mora, T. Winograd, and R. Flores, Action Workflow as the Enterprise Integration Technology. Bulletin of the Technical Committee on Data Engineering, IEEE Computer Society 16 (2) 1993.

[29] R. Medina-Mora, H. Wong, and P. Flores, The Action Workflow Approach to Workflow Management. *Proc. of 4th Conf. on Computer Supported Cooperative Work*, 1992.

[30] M. Merz, B. Liberman, K. Müller-Jones and W. Lamersdorf, Inter-organisational workflow management with mobile agents in COSM. *Proc. of the 1st Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology*, 405-420, 1996.

[31] T. J. Mowbray and R. Zahavi, *The essential CORBA systems integration using distributed objects*. John Wiley and Object Management Group, 1995.

[32] H. J. Müller, Negotiation Principles, in *Foundations of Distributed Artificial Intelligence*, eds. G. M. P. O'Hare and N. R. Jennings, Wiley Interscience, 211-229, 1996.

[33] K. L. Myers and D. E. Wilkins, Reasoning about Locations in Theory and Practice, *Computational Intelligence* **14** (2), 1998.

[34] R. Neches, R. Fikes, T. Finin, T. Gruber, R. Patil, T. Senator, and W. R. Swartout, Enabling Technology for Knowledge Sharing. *AI Magazine* **12** (3), 1991.

[35] T. J. Norman and N. R. Jennings, Generating states of cooperation between autonomous agents. *Proc. 3rd Australian Workshop on Distributed Artificial Intelligence,* Perth, Australia, 109-119, 1997.

[36] T. J. Norman, N. R. Jennings, P. Faratin, and E. H. Mamdani, Designing and implementing a multi-agent architecture for business process management. In *Intelligent Agents III* (eds. J. P. Mueller, M. J. Wooldridge and N. R. Jennings) LNAI 1193, Springer Verlag, 261-275, 1997.

[37] A. Pnueli, Specification and Development of Reactive Systems. *Information Processing 86*, Elsevier, 1986.

[38] D. G. Pruitt, *Negotiation Behaviour*. Academic Press, 1981.

[39] H. Raiffa, *The Art and Science of Negotiation*. Harvard University Press, 1982.

[40] J. S. Rosenschein and G. Zlotkin, *Rules of Encounter*. The MIT Press, 1994.

[41] A. Rubinstein, Perfect equilibrium in a bargaining model. *Econometrica* **50**, 97-109, 1982.

[42] A. Sathi and M. S. Fox, Constraint directed negotiation of resource re-allocations. In *Distributed Artificial Intelligence II* (eds. L. Gasser and M. Huhns), Pitman, 163-193, 1989.

[43] C. Sierra, P. Faratin and N. R. Jennings, A Service-Oriented Negotiation Model between Autonomous Agents. *Proc. 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World,* Ronneby, Sweden, 17-35, 1997.

[44] H. Takeda, K. Iino, and T. Nishida, Agent organization with multiple ontologies. *Int. Journal of Cooperative Information Systems* **4** (4) 321-337, 1995.

[45] K. Trammel, Workflow without Fear. *Byte* April, 1996.

[46] J. M. Tenenbaum, J. C. Weber, and T. R. Gruber, Enterprise integration: Lessons from SHADE and PACT. *Proc. 1st Int. Conf on Enterprise Integration Modelling*, 356-369, 1992.

[47] N. Vulkan and N. R. Jennings (1999) “Efficient Mechanisms for the Supply of Services in Multi-Agent Environments” *Int Journal of Decision Support Systems*.

[48] M. Wooldridge, Agent-based software engineering. *IEE Proceedings on Software Engineering* **144** (1) 26-37, 1997.

[49] M. J. Wooldridge and N. R. Jennings, Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review* **10** (2) 115-152, 1995.

[50] http://www.staffware.com/

FIGURE CAPTIONS

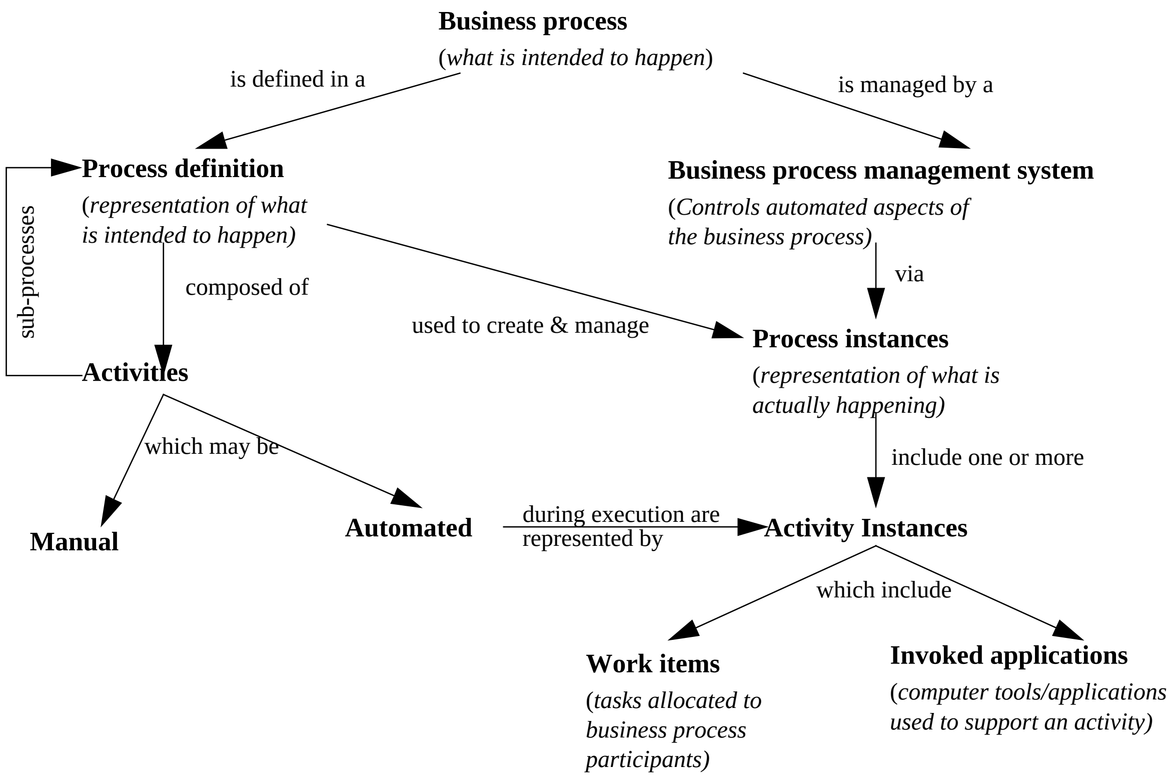

Business process
(what is intended to happen)
is defined in a
is managed by a
Process definition
(representation of what is intended to happen)
Business process management system
(Controls automated aspects of the business process)
via
sub-processes
composed of
used to create & manage
Process instances
(representation of what is actually happening)
Activities
which may be
include one or more
Manual
Automated
during execution are represented by
Activity Instances
which include
Work items
(tasks allocated to business process participants)
Invoked applications
(computer tools/applications used to support an activity)

Figure 1 : Constituent Components of a Business Process (adapted from [13]).

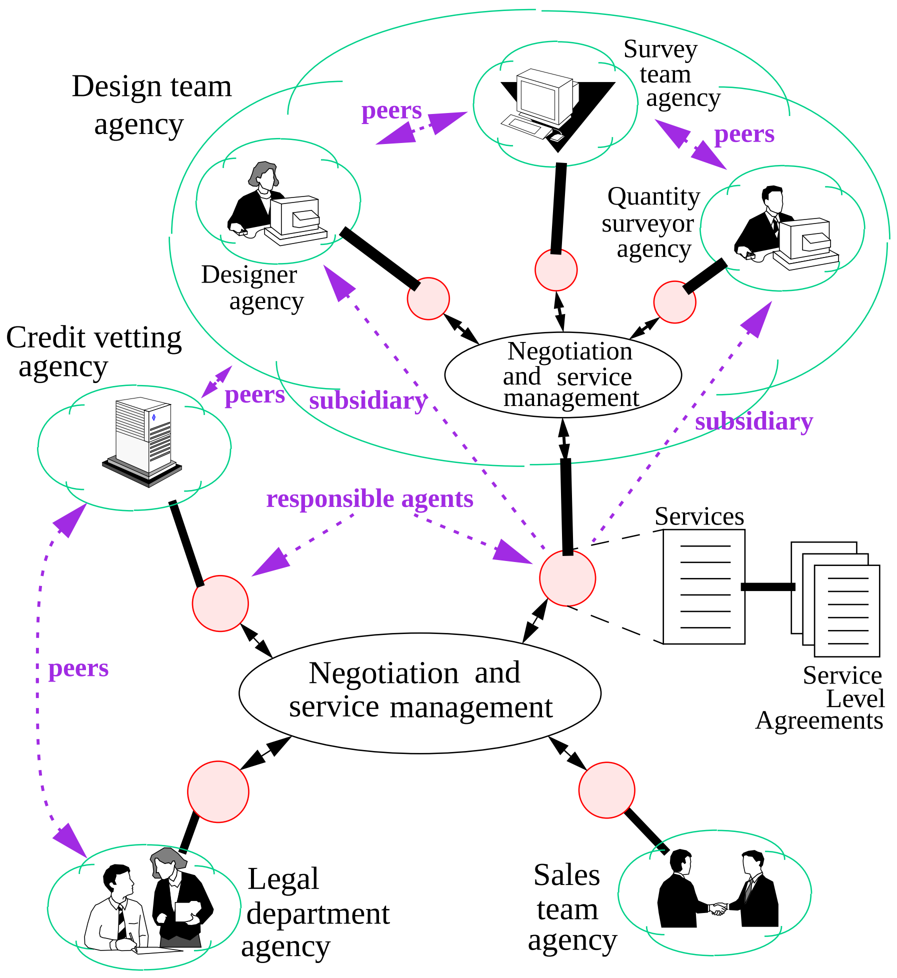


Figure 2 : The conceptual architecture of an ADEPT system.

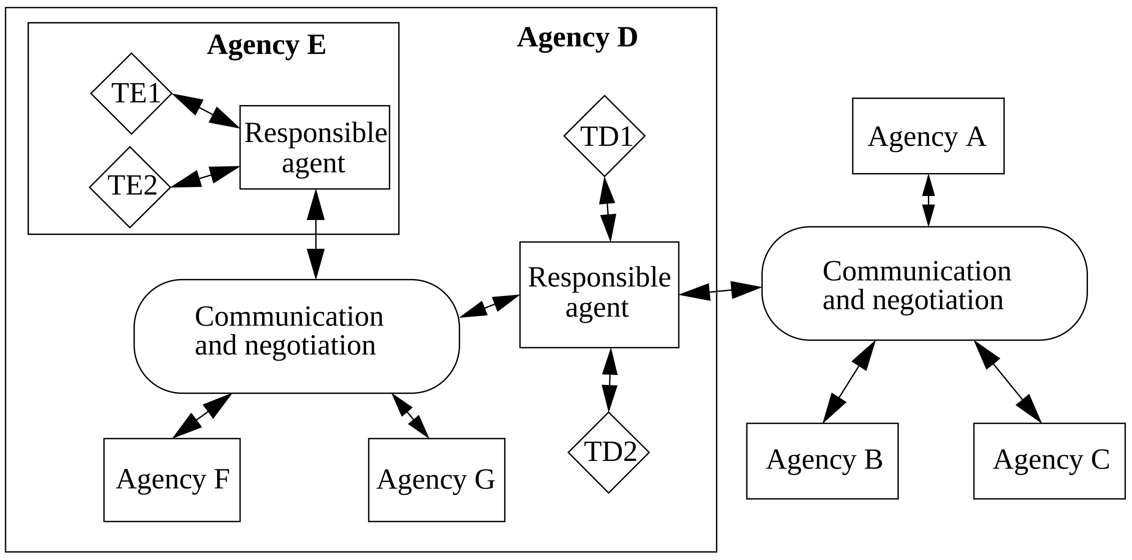
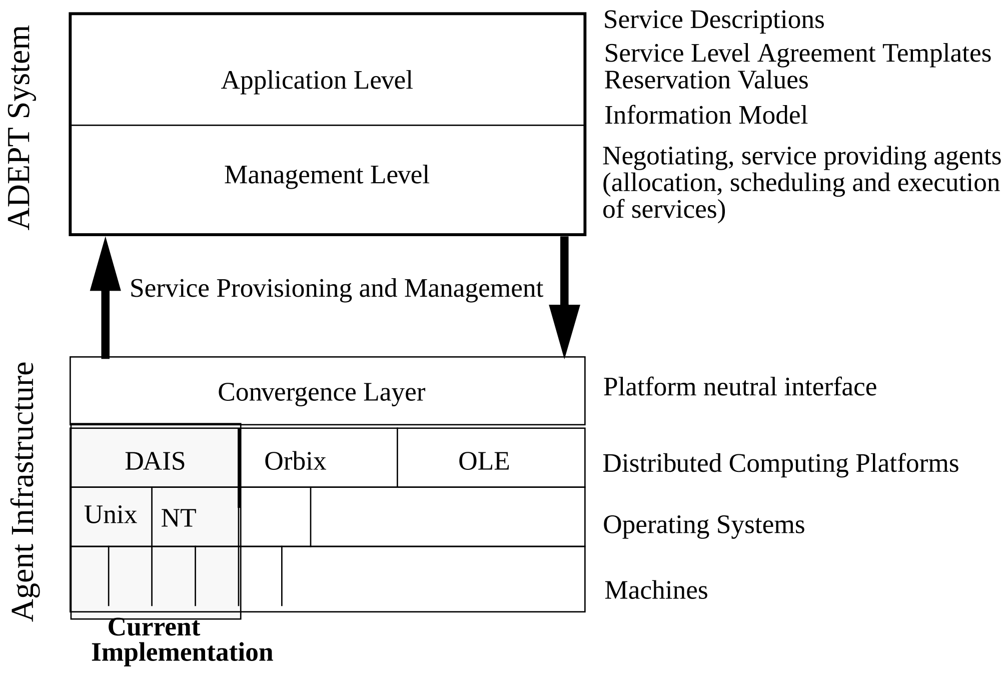


Figure 3 : The logical hierarchy of agencies



ADEPT System
Application Level
Management Level
Service Descriptions
Service Level Agreement Templates
Reservation Values
Information Model
Negotiating, service providing agents (allocation, scheduling and execution of services)
Service Provisioning and Management
Agent Infrastructure
Convergence Layer
Platform neutral interface
DAIS
Orbix
OLE
Distributed Computing Platforms
Unix
NT
Operating Systems
Machines
Current Implementation

Figure 4 : The ADEPT implementation system.

```
(service
     name       Prepare_Table
     inputs     (Home_Guests guests cli man
                 Home_TableAndChairs accommodation ser man
                 Home_Cutlery cutlery any man
                 Home_Crokery crockery any man
                 Home_Glasses glasses any opt)
     outputs    (Home_Seat_Allocation)
     guard      “( <= guests.number accommodation.number )”
     body       ( ..... ))
```

Figure 5: Exemplar Service Description: Prepare_Table

```
sequence:meal{
     must-para:organise {
          cond:have_friends “(not (empty-set service::friends))”,
          Plan_Menu(restrictions = service::friends),
          Plan_GuestList( candidates = service::friends),
          } -> (and have_friends Plan_Menu Plan_GuestList)
     can-para:prepare {
          Prepare_Food(food = Plan_Menu::menu),
          Prepare_Table(guests = Plan_GuestList::choice)
          } -> (and Prepare_Food Prepare_Table),
     Eat_Meal(<unspecified>),
     can-para:clean_up {
          Wash_Up(<unspecified>), Dry_Up(<unspecified>)
          } -> (and Wash_Up Dry_Up)
     } -> (and organise prepare Eat_Meal clean_up)
```

Figure 6: Exemplar Service Description: `Meal`

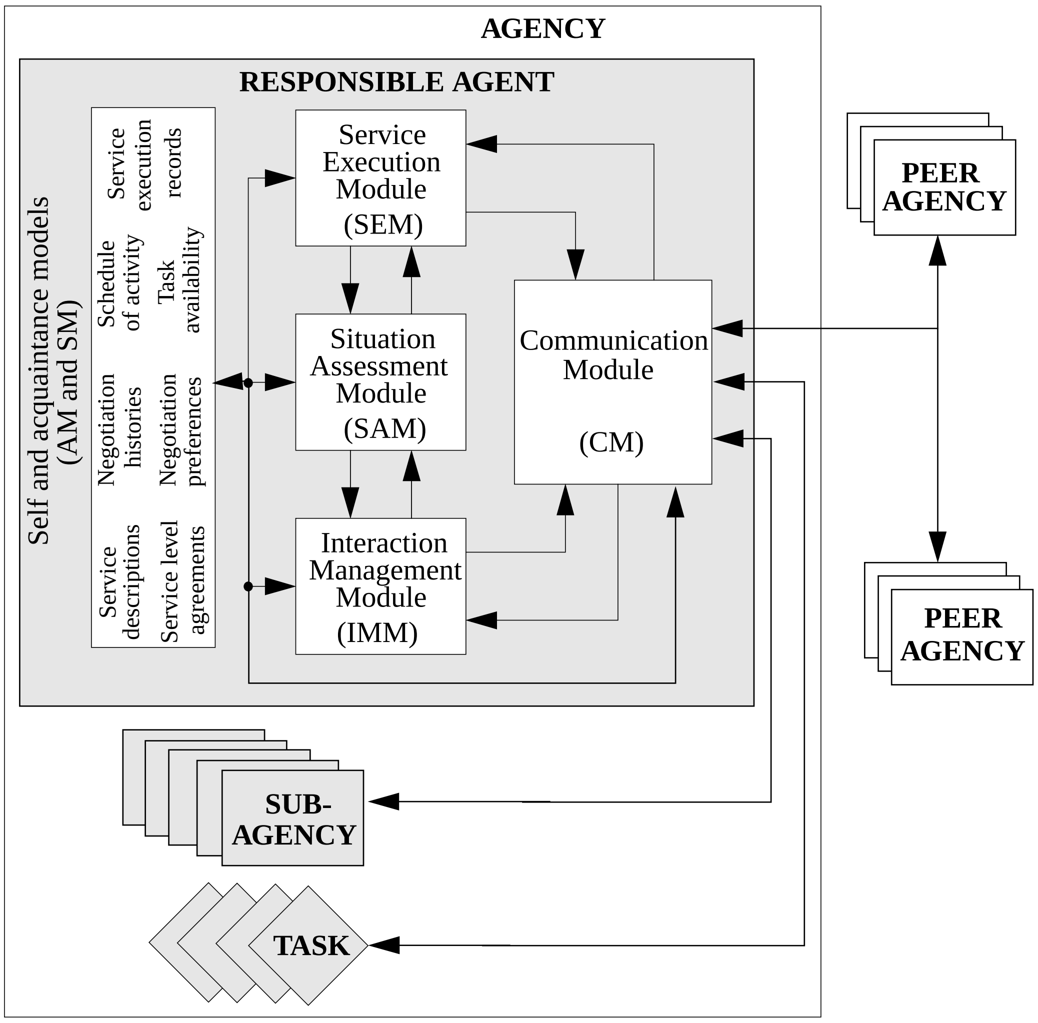

AGENCY
RESPONSIBLE AGENT
Self and acquaintance models (AM and SM)
Service execution records
Schedule of activity
Task availability
Negotiation histories
Negotiation preferences
Service descriptions
Service level agreements
Service Execution Module (SEM)
Situation Assessment Module (SAM)
Interaction Management Module (IMM)
Communication Module (CM)
PEER AGENCY
PEER AGENCY
SUB-AGENCY
TASK

Figure 7: Functional architecture of an ADEPT agent.

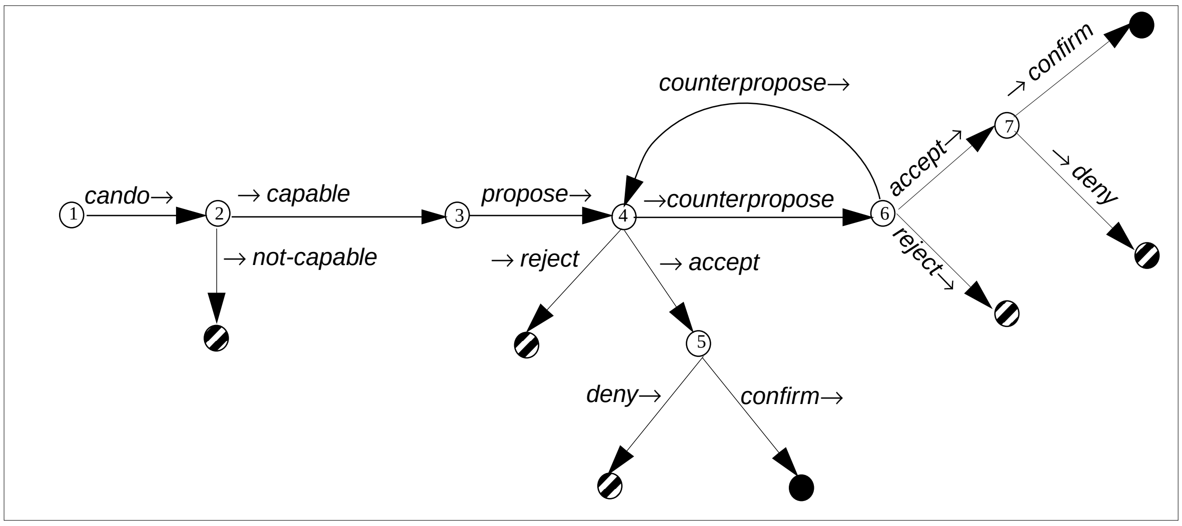


Figure 8: ADEPT's negotiation protocol. Numbered ovals represent states *during* the negotiation process and non-numbered ovals represent states associated with the *outcome* of negotiation. Shaded ovals represent unsuccessful negotiation states arrived at by the edges (primitives) `not-capable`, `reject` or `deny` and filled ovals represent successful negotiation state arrived at by the by the primitive `confirm`. Refer to table 1 for details of the individual primitives.

Slot Name	Instantiated Values
SERVICE_NAME:	cost_&_design_network
SLA_ID:	a1001
SERVER_AGENT:	NDD
CLIENT_AGENT:	CHL
SLA_DELIVERY_TYPE:	on-demand
DURATION: (minutes)	320
START_TIME:	9:00
END_TIME:	18:00
VOLUME:	35
PRICE: (per costing)	35
PENALTY:	30
CLIENT_INFO:	customer_profile
REPORTING_POLICY:	customer_quote

Figure 9: Sample service level agreement

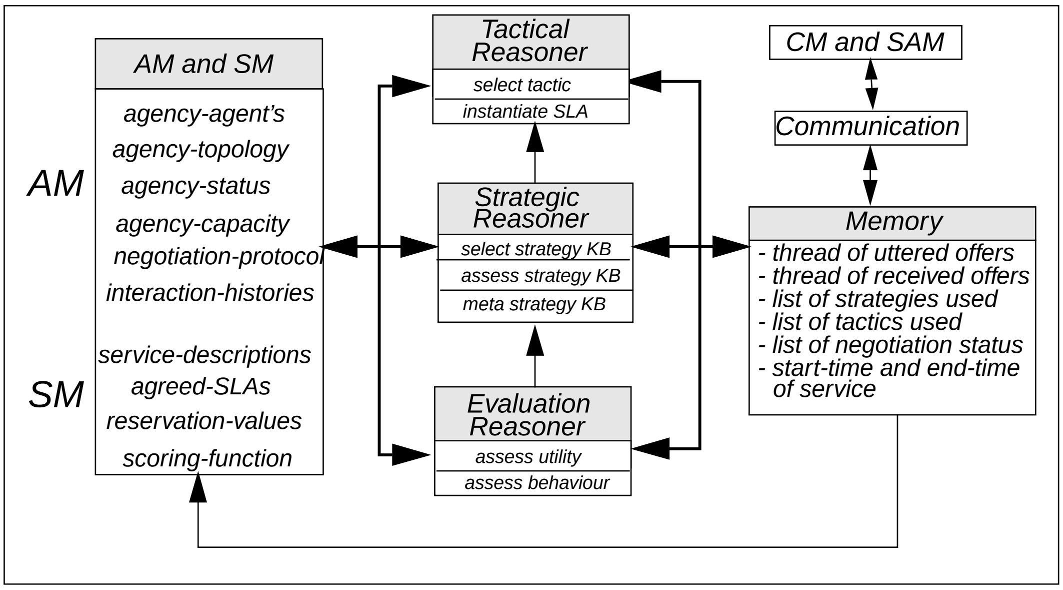


Figure 10: Internal architecture of IMM.

```
if MANY(SERVERS) & HAVE-PREFERENCE-OVER(SERVERS)
then
    NEGOTIATE(SERIAL, MOST-PREFERRED-FIRST)
```

[If there are multiple providers of a given service and the agent has developed a preference ordering over these servers for this service then negotiate with them one at a time in decreasing order of preference. This approach is adopted to minimise the amount of resource consumed since it is likely that the agent will end up choosing its most preferred acquaintance.]

```
if MANY(SERVERS) & NO-HISTORY(SERVERS)
then
    NEGOTIATE(PARALLEL)
```

[If there are multiple providers of a given service and the agent has not negotiated with any of its acquaintances for the service, then it should negotiate with them in parallel in order to build up a picture of the potential providers of this service]

```
if MANY(SERVERS) & NOW
then
    NEGOTIATE(PARALLEL)
```

[If there are multiple providers of a given service and the agent does not have much time to reach an agreement, then it should negotiate in parallel in order to maximise its chance of providing a server willing to take on the service.]

Figure 11: Negotiation Logistics Rules

```
if HAVE-TIME & DIFFERENT-ORGANISATION(OPPONENT)
then
    STRATEGY(TIME=0, RESOURCE=0, BEHAVIOUR=1)
    CONVERGENCE_SPEED(SLOW)
    LIKELY_RESPONSE(COMPETITIVE)
```

[If there is plenty of time available for negotiation and the opponent is from an external organisation, then the dominant (sole) criteria for determining the negotiation stance should be the opponent's behaviour. This will minimise the scope for being exploited, but will allow cooperative negotiation to be reciprocated. It is likely that the opponent will adopt a fairly competitive stance since the negotiation is inter-organisational. Therefore it is also likely that the negotiation will take a long time to reach a conclusion.]

```
if HAVE-TIME & SAME-ORGANISATION(OPPONENT) & MANY(SERVERS)
then
     STRATEGY(TIME=0.2,RESOURCE=0.8,BEHAVIOUR=0)
     CONVERGENCE_SPEED(FAST)
     LIKELY_RESPONSE(COOPERATIVE)
```

[If there is plenty of time for negotiation and there are multiple potential suppliers of the service (all of whom are internal to the organisation) then the important criteria is that resource usage should be minimised. A secondary consideration is that the negotiation should not take too long. Given this mixed strategy it is likely that an agreement will be reached fairly quickly and that the likely response of the negotiation opponents will be cooperative since they are all from the same organisation.]

```
if NOW & SAME-ORGANISATION(OPPONENT) & ONE(SERVER)
then
    STRATEGY(TIME=1, RESOURCE=0, BEHAVIOUR=0)
    CONVERGENCE_SPEED(FAST)
    LIKELY_RESPONSE(COOPERATIVE)
```

[If there is very little time in which to reach an agreement and there is only one potential service provider (who happens to be in the same organisation) then the dominant (sole) criteria for determining how to behave is based on the amount of time remaining. This stance is likely to result in an agreement being reached quickly and the opponent should be cooperative since it is within the same organisation.]

```
if HAVE-TIME & SAME-ORGANISATION(OPPONENT) & MANY(NEGOTIATION-THREADS)
then
    STRATEGY(TIME=0, RESOURCE=0.6, BEHAVIOUR=0.4)
    CONVERGENCE_SPEED(FAST)
    LIKELY_RESPONSE(COOPERATIVE)
```

[If there is plenty of time in which to reach an agreement for provisioning an in-house service and the agent is engaged in multiple concurrent negotiations, then the agent should ensure it takes the amount of resource consumed into account (so that it is not spending all its time negotiating) as well as the behaviour of the opponent (to reciprocate the opponent's concessions). This stance is likely to result in an agreement being reached quickly and the opponent should be cooperative since it is within the same organisation.]

Figure 12: Setting the Negotiation Strategy

```
if TIME-RUNNING-LOW & CONVERGENCE_SPEED(SLOW)&
                                           DIFFERENT-ORGANISATION(OPPONENT)
then
     STRATEGY(TIME=TIME+0.2,RESOURCE=0,BEHAVIOUR=BEHAVIOUR-0.2)
     CONVERGENCE_SPEED(UNCERTAIN)
     LIKELY_RESPONSE(UNCERTAIN)
```

[If the time by when a deal must be reached is fast approaching, the current speed of convergence is slow and the opponent is from an external organisation, then change the strategy to one which makes time remaining the dominant criterion. But pay some attention to the opponent’s behaviour to ensure that the agent is not overly exploited. In this case, it is not possible to predict how the opponent will react—it may try to stick firm and exploit the agent’s concessions or it may reciprocate the concessions in order to reach an agreement.]

```
if  HAVE-TIME & LIKELY_RESPONSE(COOPERATIVE)& EXPLOITING(OPPONENT)&
    INTENSIFYING(EXPLOITATION)
then
      STRATEGY(TIME=TIME-0.3,RESOURCE=0,BEHAVIOUR=BEHAVIOUR+0.3)
      CONVERGENCE_SPEED(LOWER)
      LIKELY_RESPONSE(COMPETITATIVE)
```

[If there is still plenty of time to reach an agreement and the current strategy should have elicited a cooperative response, but the opponent is attempting to exploit the agent (and, moreover, this tendency is increasing), then adopt a tougher stance. Base the negotiation behaviour mainly on how the opponent is behaving, but also pay some attention to the amount of time remaining. This change is likely to make the negotiation converge more slowly and may result in a more competitive stance from the opponent.]

```
if    HAVE-TIME & LIKELY_RESPONSE(COMPETITATIVE)& CONCEDING(OPPONENT)&
      DIFFERENT-ORGANISATION(OPPONENT)
then
      STRATEGY(TIME=1,RESOURCE=0,BEHAVIOUR=0)
      CONVERGENCE_SPEED(LOWER)
      LIKELY_RESPONSE(UNCERTAIN)
```

[If there is still plenty of time to reach an agreement and the current stance should have elicited a competitive response from the opponent (who is from an external organisation), but the opponent is conceding then become firm (base behaviour on time to reach an agreement) in order to attain maximum utility. This change is likely to prolong the negotiation and it is not possible to predict how the opponent will respond.]

Figure 13: Monitoring and Modifying the Negotiation Strategy

```
if    HAVE-TIME & DIFFERENT-ORGANISATION(OPPONENT)& MANY(SERVERS) &
      STRATEGY(?TIME,?RESOURCE,?BEHAVIOUR)
then
     OFFER_TIME=HIGH(INITIAL-OFFER), CONCEDE(SLOWLY)
     OFFER_RESOURCE=HIGH(INITIAL-OFFER),
                                          CONCEDE(BASED-ON-SERVER-NUMBERS)
     OFFER_BEHAVIOUR=HIGH(INITIAL-OFFER),
                           CONCEDE(BASED-ON-LAST-OFFER, LESS-THAN-OPPONENT)
     OFFER=AVERAGE((OFFER_TIME * ?TIME),
                     (OFFER_RESOURCE * ?RESOURCE)
                     (OFFER_BEHAVIOUR * ?BEHAVIOUR) )
```

[If there is plenty of time in which to reach an agreement and there are many potential suppliers of the service (all of whom are external to the organisation) then: (i) the time based tactic should suggest a high initial offer and concede slowly (since there is plenty of time available); (ii) the resource dependent tactic should suggest a high initial offer (since it is a competitive negotiation) and concede based on the number of potential suppliers who are left in the negotiation thread; and (iii) the behaviour dependent tactic should make a high initial offer (since all the opponents are external to the organisation) and it should concede slightly less readily than the opponent (to reflect its strong market position).]

```
if    HAVE-TIME & SAME-ORGANISATION(OPPONENT)& ONE(SERVER) &
      STRATEGY(?TIME,?RESOURCE,?BEHAVIOUR)
then
     OFFER_TIME=HIGH(INITIAL-OFFER), CONCEDE(SLOWLY)
     OFFER_RESOURCE=LOW(INITIAL-OFFER),
                                            CONCEDE(BASED-ON-NUM-MESSAGES)
     OFFER_BEHAVIOUR=MEDIUM(INITIAL-OFFER),
                           CONCEDE(BASED-ON-LAST-OFFER, MORE-THAN-OPPONENT)
     OFFER=AVERAGE((OFFER_TIME * ?TIME),
                     (OFFER_RESOURCE * ?RESOURCE)
                     (OFFER_BEHAVIOUR * ?BEHAVIOUR) )
```

[If there is plenty of time in which to reach an agreement and there is one potential provider who is from the same organisation then: (i) the time based tactic should suggest a high initial offer and concede slowly (since there is plenty of time); (ii) the resource dependent tactic should make a low initial offer and concede based on the number of messages sent in the negotiation thread (so the organisation's resources are not unduly wasted); and (iii) the behaviour dependent tactic should make a reasonable initial offer (since the opponent is from the same organisation) and it should concede slightly more than its opponent (in order to try and bring about a quick agreement)]

Figure 14: Negotiation Tactic Rules

Action	Content	Meaning	Context
`cando (x,y,s)`	Empty.	Sender x asks if the recipient y is, in principle, able to provide service s.	Message can be sent by any agent at any time.
`not-capable (x,y,s)`	Empty.	x informs y that it is incapable of performing s.	Used by x only in response to a `cando` action.
`capable (x,y,s)`	Empty.	x informs y that, in principle, it is capable of performing s.	Used by x only in response to a `cando` action.
`propose (x,y,s,sla)`	A single SLA information object.	x proposes to y that y performs s under the conditions specified in sla.	x must believe that y is capable of performing s.
`counterpropose (x,y,s,F)`	A nonempty list of SLA fields, F.	x proposes to y that the service provider (it may be either x or y) performs s under the conditions described in the SLA that is on the table modified with the list of fields F.	Used only in response to either an action of type `propose` or `counterpropose`.
`accept (x,y,s)`	Empty.	x accepts and commits to performing s under the SLA that is on the table.	Used in response to a `propose` or a `counterpropose`
`reject (x,y,s)`	Empty.	x rejects the SLA on the table outright, and wishes to terminate the negotiation.	Used only in response to either an action of type `propose` or `counterpropose`.
`confirm (x,y,s)`	Empty.	x commits to the SLA on the table.	Used only in response to an action `accept`.
`deny (x,y,s)`	Empty.	x withdraws its proposal from the table. (In certain circumstances this action may have consequences; agent x may have to pay some penalty to y.)	Used only in response to an `accept` action.
`renegotiate (x,y,c)`	A communicative act of type: `propose`, `accept`, `reject`, `confirm`, `deny` or `counterpropose`	x intends that the communicative action, c, should be interpreted as the negotiation message it represents, but in the context of the agent renegotiating an existing agreement.	A SLA must already exist between x and y.
`request (x,y,a)`	An instruction to the service provider; this may be to start providing the service, suspend it, terminate the service, etc.	x requests that under the agreement (indicated in the conversation identifier field of the message), y perform action a with respect to that service.	The SLA to which this message refers must exist, and the instruction must be acceptable within the scope of that agreement.
`report (x,y,r)`	A report on the state of a service being performed by x for y.	x reports to y that the state of the service execution is r.	The SLA to which this message refers must exist, and an instance of the service must be being executed by x.
`inform (x,y,i)`	One or more information objects.	x provides y with information relevant to the execution of the indicated service. If x is the consumer, i will be some input, and if x is the provider, i will be some output.	The SLA to which this message refers must exist, and the information must be relevant to the execution of that service.

Table 1: ADEPT's Agent Communication Language