

AGENT-BASED BUSINESS PROCESS MANAGEMENT

N. R. JENNINGS¹, P. FARATIN¹, M. J. JOHNSON¹, T. J. NORMAN¹, P. O'BRIEN² and M. E. WIEGAND²

¹ *Dept. Electronic Engineering, Queen Mary & Westfield College, Mile End Road, London E1 4NS, UK.*

² *BT Research Labs, Martlesham Heath, Ipswich, Suffolk IP5 7RE, UK.*

Received

Revised

This paper describes work undertaken in the **ADEPT** (Advanced Decision Environment for Process Tasks) project towards developing an agent-based infrastructure for managing business processes. We describe how the key technology of negotiating, service providing, autonomous agents was realised and demonstrate how this was applied to the BT (British Telecom) business process of providing a customer quote for network services.

Keywords: Intelligent agents; Business process management; Negotiation; Information sharing.

1. Introduction

Company managers make informed decisions based on a combination of judgement and information from marketing, sales, research, development, manufacturing and finance departments. Ideally, all relevant information should be brought together before judgement is exercised. However obtaining pertinent, consistent and up-to-date information across a large company is a complex and time consuming process. For this reason, organisations have sought to develop a number of Information Technology (IT) systems to assist with various aspects of the management of their business processes. Such systems aim to improve the way that information is gathered, managed, distributed, and presented to people in key business functions and operations. In particular, the IT system should:

- allow the decision maker to access relevant information wherever it is situated in the organisation (this should be possible despite the fact that information may be stored in many different types of system and in many different information models);
- allow the decision maker to request and obtain information management services from other departments within the organisation (and in some cases even from outside the organisation);
- proactively identify and deliver timely, relevant information which may not have been explicitly asked for (e.g. because the decision maker is unaware of its existence);
- inform the decision maker of changes which have been made elsewhere in the business process which impinge upon the current decision context;
- identify the parties who may be interested in the outcome and results of the decision making activity.

Analysis of a number of business processes, from various industrial and commercial domains, resulted in several common characteristics being identified:

- Multiple organisations are often involved in the business process. Each organisation attempts to maximise its own profit within the overall activity.
- Organisations are physically distributed. This distribution may be across one site, across a country, or even across continents. This situation is even more apparent for virtual organisations¹ which form allegiances for short periods of time and then disband when it is no longer profitable to stay together.
- 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 sub-parts are decentralised, there is often a need to place constraints on the entire process (e.g. total time, total budget, etc.).
- Business processes are highly dynamic and unpredictable—it is difficult 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 which are produced are often disrupted by unavoidable delays or unanticipated events (e.g. people are ill or tasks take longer than expected).

Given these characteristics, it was decided that the most natural way to view the business process is as a collection of autonomous, problem solving agents which interact when they have interdependencies. In this context, an agent can be viewed as an encapsulated problem solving entity which exhibits the following properties²:

- *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 artificial agents and humans in order to complete their problem solving and to help others with their activities. This requires that agents have, as a minimum, a means by which they can communicate their requirements to others and an internal mechanism for deciding what and when social interactions are appropriate (both in terms of generating requests and judging incoming requests).

- *Proactiveness*: agents take the initiative where appropriate.
- *Responsiveness*: agents perceive their environment and respond in a timely fashion to changes which occur in it.

The choice of agents as a solution technology was motivated by the following observations: (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 sub-parts 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 with which agents are endowed); and (iv) the problem solution cannot be entirely prescribed from start to finish (the problem solvers need to be responsive to changes in the environment and to unpredictability in the business process and proactively take opportunities when they arise). When taken together, this set of requirements leaves agents as the strongest solution candidate—(distributed) object systems have the encapsulation but not the sophisticated reasoning required for social interaction or proactiveness, and distributed processing systems deal with the distributed aspect of the domain but not with the autonomous nature of the components.

The remainder of this paper describes the work undertaken to conceptualise business process management as a collection of intelligent agents. Section 2 describes the key concepts of agents which offer services to one another. Section 3 details the application of ADEPT agents in British Telecom's (BT's) customer quote business process. Section 4 contrasts the ADEPT view with that of other common techniques for business process management. Finally, section 5 describes the ongoing work and the open issues which still need to be addressed.

2. The Business Process as a Community of Negotiating Agents

Each agent is able to perform one or more *services* (figure 1). A service corresponds to some unit of problem solving activity (section 2.2). The simplest service (called a *task*) represents an atomic unit of problem solving endeavour in the ADEPT system. These atomic units can be combined to form *complex services* by adding ordering constraints (e.g. two tasks can run in parallel, must run in parallel, or must run in sequence) and conditional control. The nesting of services can be arbitrarily complex and at the topmost level the entire business process can be viewed as a service.

Services are associated with one or more agents which are responsible for managing and executing them. Each service is managed by one agent, although it may involve execution of sub-services by a number of other agents. Since agents are autonomous there are no control dependencies between them; therefore, if an agent requires a service which is managed by another agent it cannot simply instruct it to start the service^a. Rather, the agents must come to a mutually acceptable agreement about the terms and conditions under which the desired service will be performed (such contracts are called *service level agreements* (SLAs)—see section 2.3). The mechanism for making SLAs is *negotiation*—a

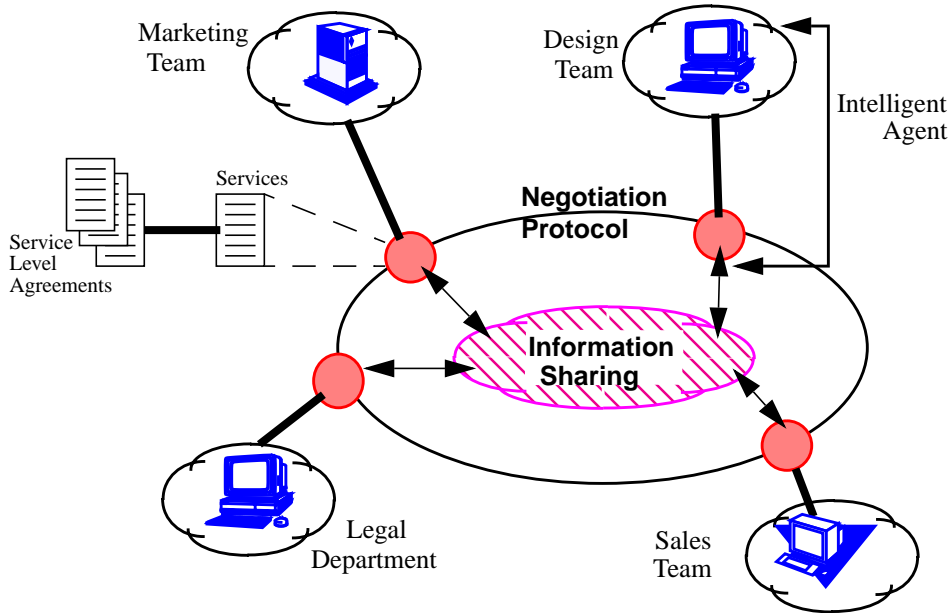


Fig. 1. An ADEPT environment.

joint decision making process in which the parties verbalise their (possibly contradictory) demands and then move towards agreement by a process of concession or search for new alternatives³.

To negotiate with one another, agents need a *protocol* which 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 organisations because each department typically models its own information and resources in its own way. 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 2.4).

2.1. The ADEPT Agent Architecture

All ADEPT agents have the same basic architecture (figure 2). This involves a *responsible agent* which interacts with peers and the *subsidiary agencies* and *tasks* within its *agency*. An agent's agency represents its domain problem solving resources. The responsible agent

^a. This is one of the major features which distinguishes multi-agent systems from more traditional forms of distributed processing⁴.

has a number of functional components concerned with each of its main activities—communication, service execution, situation assessment, and interaction management (see description below for more details). This internal architecture is broadly based on the GRATE^{5, 6} and ARCHON⁷ agent models. The domain resources can either be atomic tasks or agents representing subsidiary agencies (sub-agents). The latter case allows a nested (hierarchical) agent system to be constructed in which higher-level agents realise their functionality through lower level agents (the lower level agents have the same structure as the higher level ones and can, therefore, have sub-agents as well as tasks in their agency). For example, the higher level agent may represent a legal department whose work is carried out by a number of lawyers (the lower level agents!). This structure enables flat, hierarchical, and hybrid organisations to be modelled in a single framework^b. The differences between an agent in an agency (i.e. an agent that is responsible for a subsidiary agency) and a peer agent (i.e. an agent that is responsible for a peer agency) relate to the levels of autonomy and helpfulness. In both cases the agents negotiate to reach agreements. However in the former case: (i) the agent cannot reject the proposal outright (although it can counter propose until an acceptable agreement is reached); and (ii) the agent must negotiate in a cooperative (rather than a competitive) manner since there is a degree of commonality of purpose. In summary, there is a tight coupling between an agent and it's agency and a loose coupling between an agent and it's peers⁸.

2.1.1. *Communication Module*

The communication module routes messages between an agent and its agency and between peer agents. During task execution and management (e.g. the activation, suspension, or resumption of a task), messages are routed between the agent's Service Execution Module (SEM) (figure 2) and the tasks managed by that agent. During service execution management (e.g. the initiation of a service to be provided by another agent under some agreement, or a report of the results of a completed service), messages are routed between the agent's SEM and the SEM of another agent. During negotiation, messages are routed between the agent's Interaction Management Module (IMM) (figure 2) and the IMM of the agent or agents being negotiated with.

2.1.2. *Interaction Management Module*

The interaction management module provisions services through negotiation. The Situation Assessment Module (SAM) invokes the IMM to begin negotiation for services the agent needs. The IMM's decision making capabilities are supported by three types of information: scheduler constraints emanating from the SAM; knowledge an agent has about itself and it's own domain (represented in the Self Model (SM)); and knowledge the agent holds about peer agents (represented in the Acquaintance Model (AM)). Based on these sources of knowledge and the negotiation model (section 2.3), the IMM generates

^b. This modelling ability is important because commercial environments are founded on organisational models where an enterprise is logically divided into a collection of services. The agent-agency concept draws upon this principle to group services and tasks where it makes pragmatic sense.

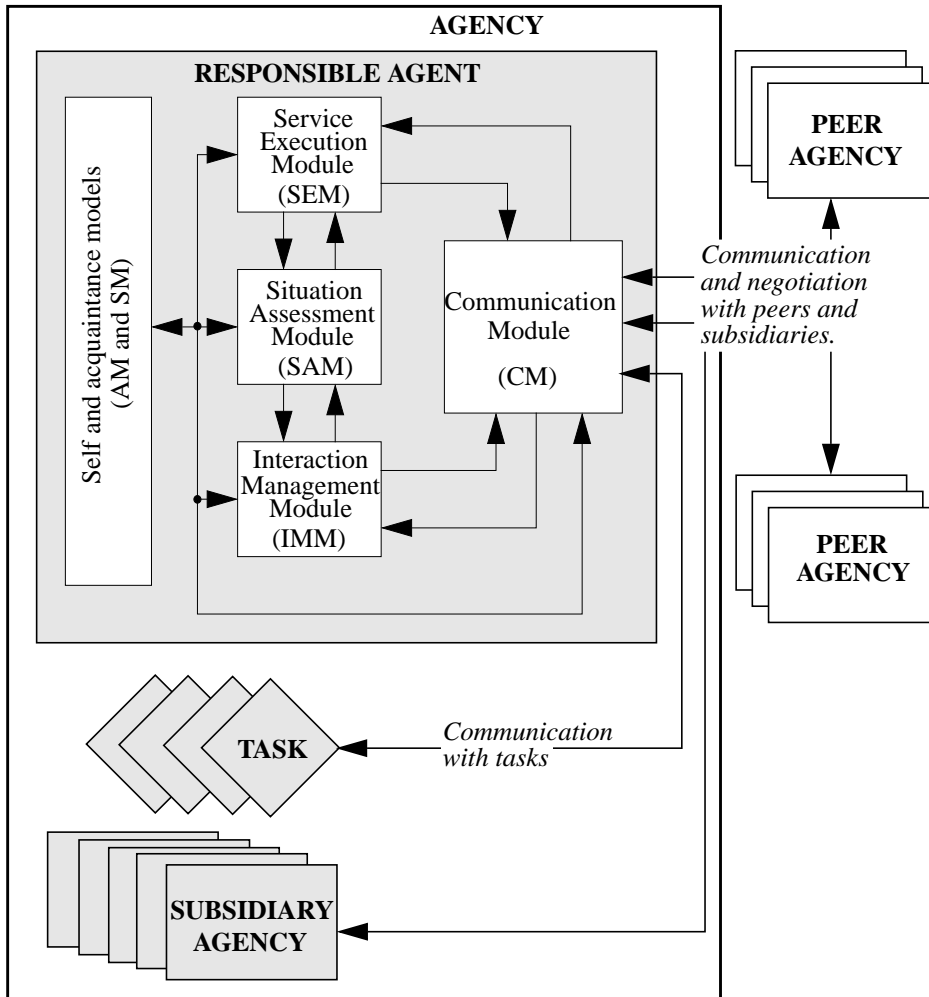


Fig. 2. The ADEPT agent architecture.

initial proposals, evaluates incoming proposals, produces counterproposals, and, finally, accepts or rejects proposals. If a proposal is accepted then the IMM creates a new SLA to represent the agreement.

2.1.3. Situation Assessment Module

The situation assessment module is responsible for assessing and monitoring the agent's ability to meet the SLAs it has already agreed and the potential SLAs which it may agree in the future. This involves two main roles: scheduling and exception handling. The former involves maintaining a record of the availability of the agent's resources which can then be used to determine whether SLAs can be met or whether new SLAs can be

accepted. The exception handler receives exception reports from the SEM during service execution (e.g. “service may fail”, “service has failed”, or “no SLA in place”) and decides upon the appropriate response. For example, if a service is delayed then the SAM may decide to locally reschedule it, to renegotiate its SLA, or to terminate it altogether.

2.1.4. *Service Execution Module*

The service execution module is responsible for managing services throughout their execution. This involves three main roles: service execution management (start executing services as specified by the agent’s SLAs), information management (routing information between tasks, services and other agents during execution), and exception handling (monitoring the execution of tasks and services for unexpected events and then reacting appropriately).

2.1.5. *Acquaintance Models*

Within the acquaintance model, the agent maintains and provides access to the SLAs agreed with other agents, and a list of peers which can provide services of interest.

2.1.6. *Self Model*

The self model is the primary storage site for SLAs to which the agent is committed, descriptions of the services the agent can provide, run time application/service specific information (e.g. the services which are currently active and the current number of invocations of each active service), and generic domain information (e.g. the upper limit the agent will pay for a service and the maximum permissible number of concurrent invocations of each service).

2.2. *The Service Lifecycle*

There are three distinct phases to the service lifecycle (figure 3). Firstly, the agent programmer has to describe the service and how it is realised. This is carried out using ADEPT’s *service description language* (SDL). As an illustration, figure 4 shows a service description from the customer quote business process (section 3). A service is described by a name, its inputs, its outputs, and its body. The name uniquely identifies the service within the particular agent in which it is situated. The input field specifies what information is needed by the service, who is to provide it, and whether it is mandatory (must be provided before the service can start) or optional (if available it will be used, but if it is unavailable the service can still proceed). In the example shown, the service must have both of its inputs available: `cr_profile` of type `Bt_CrProfile`^c from the client agent and `cust_details` of type `Bt_CustomerDetails` from the server agent. The output field specifies the information produced by the service (in this case it is `network_design` which is of type `Bt_NetworkDesign` and `detailed_reqs`

^c. The various types of information are defined in the agent’s information model. Creating an agent involves determining the information model to be used as well as specifying the service details (see section 2.4).

which is of type `Bt_CustomerReqs`). The body specifies the way the service is realised (i.e. which services and tasks need to be performed, their partial order, and the information shared between them) and the conditions which prevail if it is successful (the construct specifying this is the *completion expression*)^d. In the example shown, the mainblock of the service consists of three sub-services (`task_analyse_reqs`, `subblock` and `task_design_network`) which need to be executed in sequence. Associated with `mainblock` is a completion expression, (`and task_analyse_reqs subblock task_design_network`), which specifies that each of the sub-services must successfully complete if the whole service is to succeed.

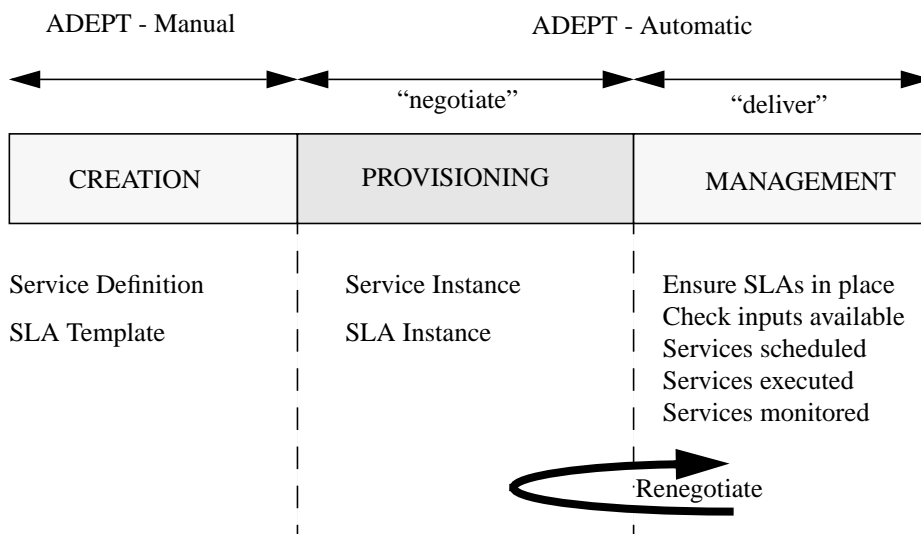


Fig. 3. The service lifecycle.

The first sub-service to be executed is `task_analyse_reqs` which takes `cr_profile` as its input and produces as its output `detailed_reqs` and `SurveyReqd`. When this sub-service finishes, the completion expression within which it was invoked is evaluated. If `task_analyse_reqs` fails (i.e. the requirements cannot be analysed) then the whole network design service is terminated since the completion expression (a conjunction) will necessarily fail. In this case, the remaining sub-services are not executed. If `task_analyse_reqs` succeeds, then the overall completion

^d. 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 the ADEPT 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.

expression is still evaluated. However, in this case it's value is unknown since although `task_analyse_reqs` is true the values of the other two services in the conjunction are unknown at this point (the conjunction of the truth values true and unknown is unknown).

```
(service
  name: Bt_DesignNetwork
  inputs: (Bt_CrProfile cr_profile client mandatory
          Bt_CustomerDetails cust_details server mandatory)
  outputs: (Bt_NetworkDesign network_design
           Bt_CustomerReqs detailed_reqs)
  body: (
    sequence: mainblock{
      task_analyse_reqs(cr_profile ?detailed_reqs ?SurveyReqd),
      sequence: subblock{
        cond:cond1(SurveyReqd = True),
        task_survey_CPE (cr_profile cust_details ?cpe_spec)
      } -> (or (not cond1) (and cond1 task_survey_CPE)),
      task_design_network
        (cr_profile cust_details detailed_reqs !cpe_spec ?network_design)
    } -> (and task_analyse_reqs subblock task_design_network)
  )
)
```

Fig. 4. A sample SDL description.

Assuming the requirements are successfully analysed, the next sub-service is executed. Subblock is a composite construct involving two sequential actions. The first component is a conditional statement which must evaluate to true before the second component (`task_survey_CPE`) is performed. The conditional tests whether a survey is required. If a survey is not needed (i.e. the conditional is false) then the completion expression in the subblock is satisfied since `(not cond1)` is true (the completion expression is a disjunction). Control then switches to `task_design_network`. Alternatively, if a survey is needed then `cond1` is true and hence the completion expression's first disjunct is false. Since the second disjunct remains unknown, subblock does not fail at this point. Subblock's second sub-service, `task_survey_CPE`, is then executed. This service takes two inputs—`cr_profile` and `cust_details`—and produces `cpe_spec`. If `task_survey_CPE` is successful, the second disjunct in subblock's completion expression is satisfied which means that subblock succeeds.

If subblock succeeds, `task_design_network` is executed. This takes as its input `cr_profile`, `cust_details`, `detailed_reqs` and `cpe_spec` (optional input) and produces as its output `network_design`. If this sub-service completes then mainblock completes since the conjunction of the three sub-services is now true.

Once a service has been created and placed within an agent it becomes accessible to the other agents in the system. To activate a service, the client and the server agents negotiate until they come to a mutually acceptable SLA—*no service can be executed without a concomitant SLA being in place*. An important facet of this negotiation is the manner in which the service is provisioned. ADEPT supports three different provisioning 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) *Regular*: the service is required a number of times, but it is known in advance when it is needed; and (iii) *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 specified in the SLA).

If the provisioning phase is successful, a specific instance of the service is created for execution within the context of an associated SLA instance. At some point the agent needs to execute the service, this requires it to ensure: that appropriate SLAs are in place for constituent sub-services, that the required input information is available, that the service is scheduled so that any constraints specified in the SLA are met, and, ultimately, that the appropriate services and tasks are executed (either within the local agency or by the chosen peer agent). Since the agent is situated in a dynamic and unpredictable environment, it must keep track of its context—thus new services may be agreed which require the agent to reschedule its resources or currently scheduled services may fail and require the agent to replan its execution strategy. In the extreme case, the agent may even need to return to the provisioning phase to renegotiate a SLA which cannot be satisfied in the current situation (see figure 3).

2.3. The Negotiation Model

There are three components of the ADEPT negotiation model—the protocol, the service level agreements, and the reasoning model. The protocol covers the process of finding out the services an agent can perform (agent sends out a CAN-DO primitive and respondents return a I-CAN primitive), the provisioning phase of coming to an agreement (PROPOSE, COUNTER-PROPOSE, ACCEPT, and REJECT), and the management phase of actually invoking the agreement (i.e. instructing agents to activate, suspend or resume a service, and informing agents of completions or failures of a service); see Alty et al.⁹ for more details of this work. The novel aspects of negotiation in the ADEPT system relate to the types of agreements which agents can make and the models they use to guide their negotiation behaviour. The requirements of the business process domain mean that agreements need to be more encompassing and the reasoning more elaborate than those found in most extant multi-agent systems.

The nature and scope of the SLAs are derived almost exactly from the types of legal contract which are often used to regulate current business transactions (figure 5 is a typical example taken from the BT customer quote business process management system illustrated in figure 10). *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

who are party to the agreement. `Delivery_type` identifies the way in which the service is to be provisioned (section 2.2). The SLA's scheduling information is used by the SAM and the SEM for service execution and management—`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 CSD (i.e. the agent representing the customer services department) can invoke agent DD (i.e. the agent representing the design department) to cost and design a customer 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. `Client_info` specifies the information the client must provide to the server at service invocation (in this case CSD must provide the customer profile) and `reporting_policy` specifies the information the server returns upon completion.

Slot Name	Instantiated Values
SERVICE_NAME:	cost_&_design_customer_network
SLA_ID:	a1001
SERVER_AGENT:	DD
CLIENT_AGENT:	CSD
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:	cr_profile
REPORTING_POLICY:	customer_quote

Fig. 5. Exemplar service level agreement

The reasoning model also represents a novel contribution of this work. Existing work on negotiation can be divided into two distinct camps. Theoretical work^{10, 11, 12} provides important insights into how agents should negotiate to produce optimal solutions. However, a number of unrealistic assumptions are common in these negotiation models; typical assumptions include the availability of complete action descriptions, a utility function that can order all alternatives in all contexts, and that agents exhibit perfect rationality

when selecting actions. In contrast, the practical work typically adopts a very superficial approach to negotiation. In the much vaunted contract net protocol⁴, for instance, a manager sends out a request to a number of potential contractors to provide a given service to a given degree of quality. The potential contractors return a bid if they are capable of fulfilling all the requirements. The manager then selects the best bid. This model fails to capture many intuitive and important aspects of the negotiation process. For example, bidders cannot counter-propose better options, they cannot modify any of the service agreement parameters, and the emphasis in devising a complete specification is placed solely with the task manager.

The approach within ADEPT is to develop a deep and explicit model of the process of negotiation (this terminology is analogous to its use in the context of reasoning models for second generation expert systems^{13, 14, 15}). Such a model is needed to capture the richness of the interactions which take place when setting up agreements in this domain. The model covers the whole process of generating initial offers, evaluating offers, and counter proposing if offers are unacceptable.

The model has two component knowledge bases: a declarative one and a procedural one. The former, represented as a causal network, explicitly models what is being negotiated for and why the negotiation is taking place (i.e. it sets the negotiation context). For example, negotiation over the price of a service is a meta-service conflict that can be caused by an agent believing it is being over charged. Similarly, an agent may need to negotiate over a service's start time if the client's proposal conflicts with it's existing commitments. The procedural knowledge base, represented as a set of strategies and mechanisms for selecting between them, specifies which actions should be taken given the declarative knowledge. For example, given that the agent needs to negotiate over price, the knowledge base may indicate that Boulwarism^e is a good strategy to adopt if the agent has a long time to reach an agreement or if there are many potential suppliers of the service. In such cases, the agent generates a price offer and continues to counter-propose that initial offer throughout the negotiation. Alternatively, if the agent wants to reach an agreement for a scarce service or if it is negotiating with an agent in its agency, then it may adopt the more cooperative tit-for-tat strategy—making concessions when the agent concedes and standing firm when the other agent is uncompromising.

2.4. Information Sharing

Agent negotiation requires a reliable means of communication. Such communication can be viewed on two levels: (i) actually transporting the messages; and (ii) conveying the desired meaning of the message. The former is handled transparently by the agent's underlying infrastructure (see section 2.5). The latter is more problematic and requires conceptual design. Because of the characteristics of the business process domain (section 1), it is impractical to insist that all agents conform to a common model of information. For example, a surveyor may find it necessary to represent location information in

^e. Boulwarism is the strategy in which the negotiator makes a reasonable initial offer and then sticks firm throughout the negotiation¹¹.

terms of grid references on a particular map, but within the customer liaison department a location may be described in terms of an address. If autonomous agents representing these two departments are to communicate such information, they must be aware of the differences in their models of information.

In common with a number of existing approaches to software agent interoperation (e.g. the SHADE and PACT projects¹⁶ and the Knowledgeable Community project¹⁷), for an agent to participate in an ADEPT environment it is necessary for it to communicate using a common expressive language. This common language consists of a protocol and a syntax for expressing information. Furthermore, for one agent to understand the meaning of another's communicative actions they must either share a common information model or have the ability to transform information between their respective models. This enables agents within the community to: (i) know the *intention* of a communicating agent via a set of illocutionary acts^{18, 19} (e.g. is the message intended to inform, deny, or request some proposition); (ii) *interpret* the contents of the message through the use of a common syntax (e.g. the Knowledge Interchange Format, KIF, which is a prefix version of first order predicate calculus with certain extensions designed for this purpose²⁰); and (iii) *understand* what the contents of the message mean via the use of a well defined vocabulary (e.g. what is it that the agent is being informed of, that is being denied, or requested). These three aspects of agent communication (expanded on below) enable agents to communicate their beliefs, goals and other mental states, and thereby negotiate and perform other complex communicative functions.

An agent operating in an ADEPT environment must use one of a set of illocutionary acts (henceforth communicative acts) to specify what the agent *intends* by the message¹⁹. Consider the action of proposing an SLA to another agent during negotiation through the use of the PROPOSE message type (see section 2.3). This indicates that the agent sending the message intends the contents of the message to be interpreted as a proposal for the provision of a particular service that the recipient has registered as being able to provide. Suppose that the agent DD (i.e. the Design Department agent illustrated in figure 10) has agreed to provide the `cost_&_design_customer_network` service. To fulfil this agreement, DD needs to survey the proposed network site, but is unable to perform this task. Therefore, the DD agent must negotiate with the SD agent (i.e. the agent representing the Surveyor Department, figure 10) to provide this service. To initiate the negotiation, DD may PROPOSE that SD provides this service at the most convenient time and cost for DD. This may, if SD is willing to negotiate, produce a response from SD; for example, SD may COUNTER-PROPOSE a later time and a higher price, or ACCEPT DD's proposal. At the head of each message an ADEPT agent must specify the intention of the message.

As well as understanding the intention behind the message, a recipient must be able to *interpret* the contents of a message for it to be understood. For example, if agent x wishes to inform agent y that it believes p , the expression contained in the message may be *believes-that*(x, p) or x *believes-that* p . The former using a prefix notation and the latter an infix notation for the operator *believes-that*. For agent y to be able to interpret the contents of a message from x they must agree on a common syntax for the representation of communicated knowledge. Therefore, either each agent must use the same syntax to commu-

nicate information, or the syntax used must be explicitly stated in the message. The Knowledge Query and Manipulation Language (KQML)²² provides a “language” slot within a message in which the content language can be specified, e.g. KIF or Prolog. In ADEPT all agents use a KIF-like content language, and so no such language slot is included in ADEPT messages (see figure 7).

The final requirement for effective inter-agent communication is that the agent receiving a message must be able to *understand* the intended meaning of the symbols and relations used in the content of the message. Suppose that agent DD has a SLA with agent SD, in which SD provides DD with a `survey_customer_site` service under certain conditions (figure 10). Then if agent DD instructs agent SD to, on the basis of their agreement, survey a particular location, agent SD must know what is meant by a “location” for it to understand the message. For an agent to be certain that the content of a message will be understood correctly by its recipient, the sender must use a vocabulary (or information model)^f that can be understood by the recipient. Suppose that in DD’s local information model, the symbol “location” refers to the postal address of the client requiring a network and the symbol “site” refers to the location of the proposed network, and that in SD’s local information model, the symbol “location” refers to the grid reference of the network site to be surveyed. These agents do not share the same model of information; they are similar, but use different symbols for the same concept, and the same symbol for different concepts for example. For agent DD to arrange a survey of the proposed site, these heterogeneous agents must communicate, and hence information that should be interpreted in terms of DD’s information model must be translated into equivalent information that may be interpreted in terms of SD’s model. An ADEPT agent is required to declare the information models in which it may represent knowledge. In this way an agent that wishes to communicate with that agent must use an acceptable information model for expressing the message content.

The ADEPT system supports the development of information models through an information modelling language. This language is object-oriented and provides the normal string, integer, float and boolean types as well as constructs such as enumerated types, object classes (similar to structures in C) and lists, which enable more complex types to be defined. For example, the structure of an SLA is defined in the ADEPT information model using this language. This information model inherits the four basic types (i.e. `Types_String`, `Types_Integer`, `Types_Float` and `Types_Boolean`) from the `Types` information model. These basic types are used, for example, to construct a model of time so that durations, start times and end times of services may be defined (see figure 6). A construct of type `Adept_Time` consists of four slots each of type `Types_Integer` that denote the day, month, hour and minute of that point in time or interval of time. A construct of type `Adept_Sla` is then defined using this type and others.

^f. An information model is a specification of the symbols that an agent uses in the content of the message. For example, it may specify that the symbol *believes-that* is a belief operator that states that the agent specified believes the proposition specified. This proposition may itself be constructed from other symbols from the information model.

```

(class Adept_Time
  (Types_Integer day)
  (Types_Integer month)
  (Types_Integer hour)
  (Types_Integer minute)
)

```

Fig. 6. The ADEPT information model for time.

The information sharing language is still under development, but will consist of a set of communicative acts PROPOSE, COUNTER-PROPOSE, ACCEPT, etc., the semantics of which are in the process of being specified, and a KIF-like²⁰ syntax. An agent must transform the information that is being communicated from its local representation into the common form before it is transmitted to the intended recipient in the `content` field of an inter-agent message. A single message (figure 7) consists of a performative specifying the agent's intention, four fields specifying the identity of the sender, recipient, the strand of conversation that the message is a part of (enabling multiple conversations to be undertaken concurrently) and the service that it concerns (a single agent may be able to provide or may require many different services). Furthermore, the recipient must be able to understand the meaning of the symbols contained in the message, and so the content of the message must be understood with reference to a specific information model, identified in the `info-model` field; each information model has a unique identifier.

```

(message
  (comm: <performative>)
  (sender: <agent-id>)
  (recipient: <agent-id>)
  (conversation: <conversation-id>)
  (service: <service-name>)
  (info-model: <model-id>)
  (content: <expression>)
)

```

Fig. 7. The general structure of an inter-agent message.

2.5. The ADEPT Implementation

ADEPT agents are implemented on top of a CORBA (Common Object Request Broker Architecture) compliant distribution platform²³ (figure 8). The platform allows transparent access to distributed objects over a heterogeneous network of machines and operating systems. The particular platform used in this work is DAIS²⁴. So that the agent system is not tied to a specific distribution platform, a layer of abstraction is placed between the agent level and the distribution platform. This *convergence layer* presents a standard interface to the agent system whatever the underlying platform. Thus DAIS could be replaced

by a comparable platform simply by devising a set of mappings from the convergence layer to that platform.

The agents are integrated with the DAIS platform by registering their CM as a DAIS object. The IMM and CM are written as rule-based systems using CLIPS²⁵ (C Language Integrated Production System). The SEM, SAM, AM and SM are written using the CLIPS object-oriented system.

The programmer of an ADEPT system does not need to be aware of these details. He specifies the agents in more abstract terms—SDL, SLAs, and information models. The agents themselves then take charge of managing the application and performing the necessary service provisioning and service management activities.

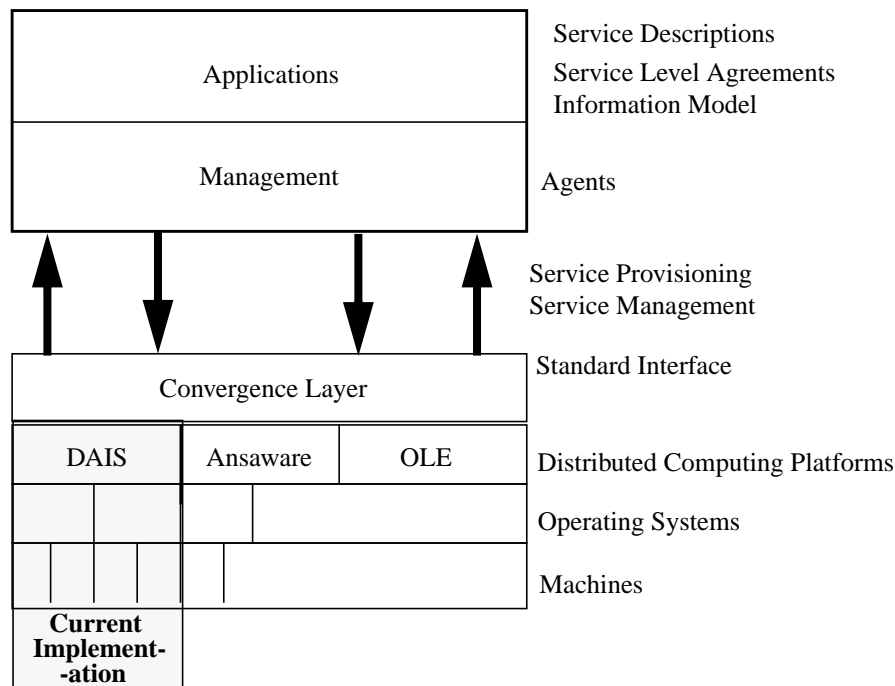


Fig. 8. ADEPT implementation system.

3. BT's Customer Quote Business Process

This scenario is based on BT's business process of providing a quotation for designing a network to provide particular services to a customer (figure 9).⁸ The process receives a

⁸ The scenario has been simplified for the purposes of explanation and demonstration. The real business process for this service contains 38 tasks and 9 choice points. Despite this simplification, the key aspects of the process are still present. Each activity requires resourcing and has a start/end point whereby progress can be measured. Choice points indicate which sequences of activities require provisioning. There are a number of concurrent activities which require coordination.

customer service request as its input and generates as its output a quote specifying how much it would cost to build a network to realise that service. It involves up to six parties: the sales department, the customer service division, the legal department, the design division, the surveyor department, and the provider of an out-sourced service for vetting customers.

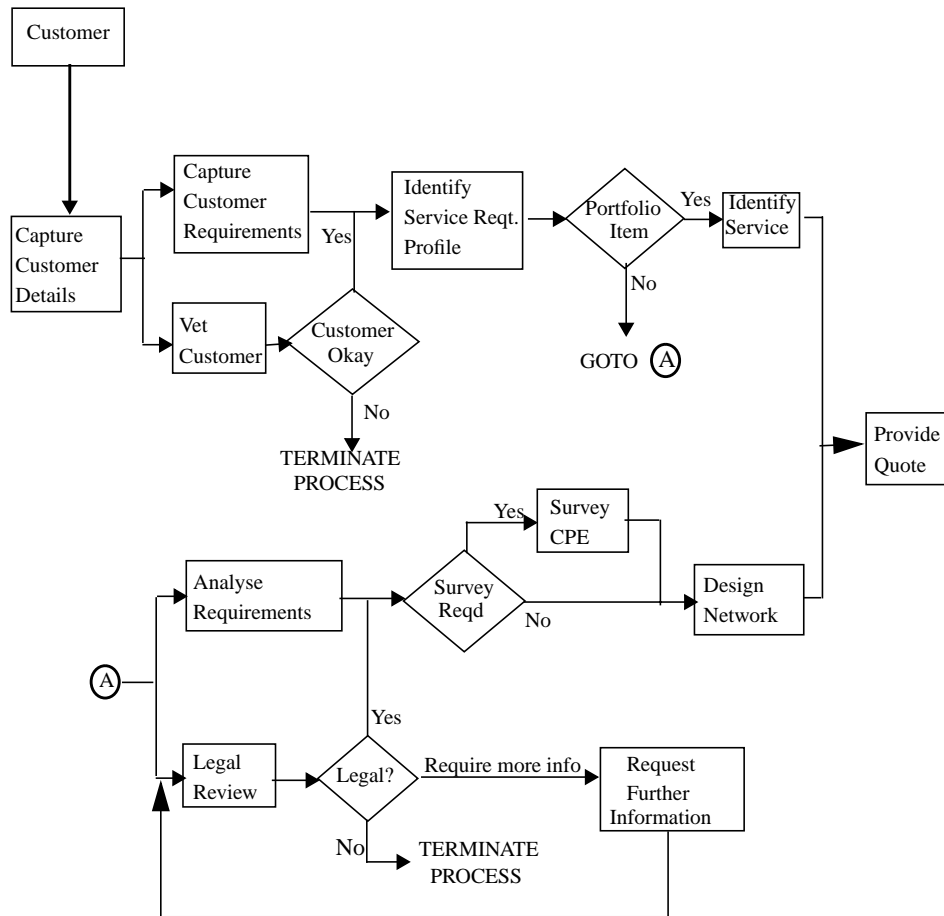


Fig. 9. The provide customer quote service.

The process is initiated by a customer contacting the customer service division. The customer's details are captured and whilst the customer is being vetted (in terms of its credit worthiness, false ID, etc.) their requirements are elicited. If the customer fails the vetting procedure, then the quote process terminates. Assuming the customer is satisfactory, it's requirements are recorded and mapped against the service portfolio. If the requirements can be met by a standard off-the-shelf portfolio item then an immediate quote can be offered based on previous examples. In the case of bespoke services, how-

ever, the process is more complex and involves a *bid manager*. The bid manager further analyses the customer's requirements and whilst this is occurring the legal department checks the legality of the proposed service (e.g. it is illegal to send unauthorised encrypted messages across France). If the desired service is illegal, then the entire quote process terminates and the customer is informed. If there is any uncertainty about the service's legality, then the business process is suspended while further information is obtained from the customer. If the requested service is definitely legal then the design phase can start. To prepare a network design it is usually necessary to have a detailed plan of the existing equipment at the customer's premises (CPE)—the exception to this is when the desired service is sufficiently simple that a survey is not warranted. Sometimes such plans might not exist and sometimes they may be out of date. In either case, the bid manager determines whether the customer site(s) should be surveyed. On completion of the network design and costing, the customer is informed of the service quote and the business process terminates.

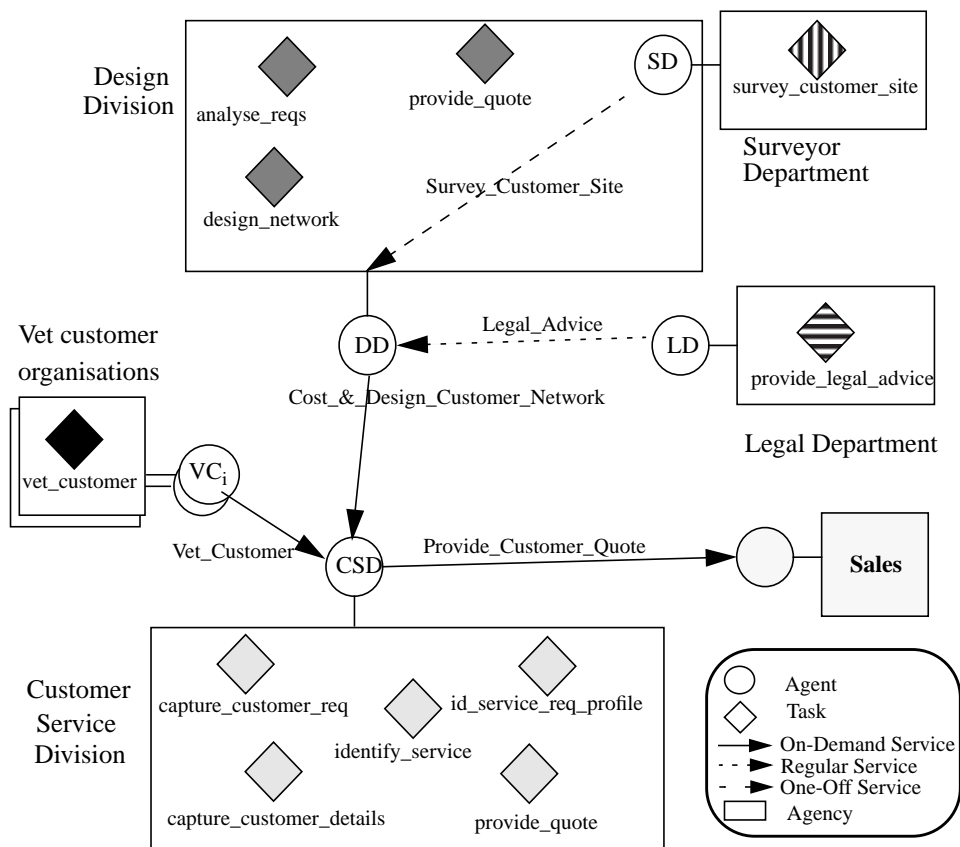


Fig. 10. Agent system for the provide customer quote business process.

From the business process description, the following agent system was designed (figure 10). The agents (denoted by the circles) were chosen to represent distinct departments or enterprises involved in the customer quote business process. The VC agents (i.e. agents representing Vet Customer agencies) represent the concerns of external enterprises as this activity is outsourced. Agent SD is within DD's agency because the design division has overall management responsibility for the surveyor department (i.e. all requests for site surveys must be channelled through the design department).

The process is triggered when the sales agent sends a request to the CSD agent (i.e. the agent representing the Customer Services Department) to provide a customer quote.^h The CSD agent identifies the SLA associated with the request: in this case it relates to the `Provide_Customer_Quote` service. The SDL body of this service is parsed to create a tree of possible routes that the SEM can take. A depth first path is selected and the tasks and services in that path are scheduled and resourced (by the SAM). The SEM begins executing the constituent sub-services and tasks. One of the first sub-services it encounters is to vet the customer (this occurs in parallel with the `capture_customer_req` task and after `capture_customer_details`). When the SEM comes to execute this service it realises (by checking it's SM) there is no associated SLA and so it reports an exception to the SAM.ⁱ The SAM determines that the service cannot be realised locally (by referring to its SM) and so it must be bought in from an external agent. It also decides that the service should be provisioned in an on-demand manner because it is an activity which is needed on each invocation of the business process. As such, it is preferable to negotiate for a longer term SLA covering multiple invocations rather than negotiating for one each time the business process is invoked. In addition to identifying the service name and the desired provisioning mode, the SAM indicates any scheduler information which influences the service's provisioning (e.g. the service's earliest start and latest end times).

Vet customer service provisioning begins with the CSD's IMM sending `CAN-DOs` to all the agents it can identify (using its AM) as being potentially able to provide this service (in this scenario there are three such agents— VC_1 , VC_2 and VC_3). Negotiation proper begins when the CSD agent concurrently sends out initial proposals (in the form of instantiated SLAs) to all the vet customer agents which responded with `I-CAN`. This initial proposal may be acceptable to one of the VC agents in which case an agreement is made and the negotiation is terminated. However, in most cases the VC agents find some part of the proposal unsatisfactory and so return a revised counterproposal to CSD. The CSD and VC agents then engage in several concurrent rounds of exchanging SLA messages until either the CSD comes to an agreement with one of the VC agents or all the VC agents reject all the offers and break off negotiation. If the CSD agent receives more than one acceptable offer, it selects the one closest to it's specified optimum^j. The chosen agent is informed of

^h. The scenario assumes an on-demand SLA between the CSD and the sales department has already been negotiated.

ⁱ. In future versions of the system, agents will pre-parse the SDL body to see which SLAs need to be set up *before* it comes to their actual execution. This will allow the agents to proactively negotiate SLAs. In this case, agents need to strike a balance between expending resources provisioning services which may not be used and only provisioning services when they are actually needed (which may delay their execution).

it's success and an SLA for the `Vet_Customer` service comes into force. The CSD agent's IMM then informs its SAM which, in turn, reinvokes the SEM's execution of `Provide_Customer_Quote` with the freshly agreed `Vet_Customer` SLA stored in its SM. Since the agreement is for on-demand provisioning—the CSD agent can ask the chosen VC agent to vet customers as and when new customers are presented to it from the sales department. The CSD agent's SEM sends a `SERVICE-ACTIVATE` request to the SEM of the selected VC agent within the time frame specified in the SLA. When the customer has been vetted, the client VC agent informs the CSD agent's SEM of the result (as specified by the SLA's reporting policy). If the customer fails the vetting procedure then `Provide_Customer_Quote` fails and the sales department is informed. If the customer is successfully vetted, the CSD agent's SEM starts executing the next sub-service.

The next sub-service checks whether the customer's request is for a portfolio item—achieved by executing the `id_service_req_profile` task. If it is a portfolio item then the service is identified (`identify_service`) and a quote is looked up (`provide_quote`) and returned to the sales department (as specified in the SLA between the CSD and the sales department). Execution of `Provide_Customer_Quote` then terminates.

If the desired service is bespoke—`id_service_req_profile` fails—then the next sub-service to be executed is `Cost_&_Design_Customer_Network`. Again the SEM informs the SAM that there is no associated SLA in place. The SAM decides the service must be bought in (after examining its SM) and that it should be provisioned in an on-demand manner (because it is required every time a customer requests a bespoke service. A one-off SLA would be justified if a significant proportion of the customer service requests were for portfolio items). It then asks the IMM to obtain an appropriate agreement. The IMM notes from its AM that the only agent offering this service is DD and so it starts negotiating with it. Assuming the two agents reach an agreement, the CSD agent's IMM informs it's SAM which informs it's SEM that an appropriate SLA is now in place (see figure 5). When the CSD agent's SEM indicates that the `Cost_&_Design_Customer_Network` service should be invoked, the DD agent starts executing it under the newly agreed SLA. This service involves executing the `Bt_DesignNetwork` sub-service (figure 4) to produce the network design, ensuring the necessary legal checks are performed, and executing the `provide_quote` task to cost the design. When the customer's requirements have been analysed in more detail (i.e. `detailed_reqs` are available from `Bt_DesignNetwork`), the legality of the customer's request is checked.^k The DD agent (i.e. the agent representing the Design Department) realises (by checking its AM) this service can only be provided by the LD agent

^j. If the negotiation fails to find any agents willing to vet customers, the CSD agent's SAM is informed and that particular invocation of `Provide_Customer_Quote` fails. In this case, service failure is reported back to the sales department. For the future, we are investigating techniques for dynamically revising the business process in such situations.

^k. Checking is managed in `Cost_&_Design_Customer_Network` by having a completion expression which either stops or suspends the design activity if the customer's request is not legal or allows it to continue if the request is legal.

(i.e. the agent representing the Legal Department) and so it starts to negotiate with it. The service is provisioned in a one-off manner because it too expensive to have waiting idle when there are no designs to check. When the agreed legal service is invoked, the requirements are checked and the appropriate course of action is taken depending on the outcome of this review.

As part of `Bt_DesignNetwork`, a survey of the customer's premises may be needed. If this is the case, the DD agent's SEM informs it's SAM that no SLA is in place for `task_survey_CPE`. The SAM notes (by examining its AM) that an agent (SD) within it's agency can provide the service. It decides the service should be provisioned in a one-off manner (because the service is only occasionally required) and so the DD's IMM negotiates with SD. Assuming they reach an agreement, the DD agent invokes the agreement and requests SD to obtain a survey for the customer's premises. When the survey is complete or after the service is declared legal if no survey is required, the network design is carried out and then costed. The service's cost is returned to the CSD agent as specified in the `Cost_&Design_Customer_Network SLA` (figure 5). The `Provide_Customer_Quote` service then completes and the quote is returned to the sales department as specified in the SLA with the sales department.

For subsequent service quote requests, several of the basic agreements for managing the business process are already in situ. The CSD agent has an on-demand SLA for vetting customers and it may also have an agreement for costing and designing the customer's network. This means there is less of a negotiation overhead on subsequent process invocations. The services which may generate further negotiations in subsequent quote processes are those which are only occasionally invoked—legal services and survey customer site.

4. Related Work

Few (if any) commercial systems exist which support business process management using agent technology. Existing business process or 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 should be 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 far 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 a workflow engine which executes business tasks in a predefined order specified in a workflow script²⁶. In recent years, they have proven their worth primarily by automating transaction intensive processes which are often found in fields such as banking²⁷.

The ADEPT system exhibits most of the characteristics of existing workflow management systems. In ADEPT, this workflow enactment function is performed by the Service Execution Module. However, unlike workflow management systems, ADEPT also performs both resource management and sophisticated exception handling:

- **Resource management:** In ADEPT, the agents have the ability to perform direct management of resources: the systems, databases, equipment and people that make up an organisation. Traditional workflow and business process management systems do not provide an inbuilt capability for direct resource management, instead processes have to be resourced and dimensioned prior to enactment. This means the ADEPT system is far more responsive to unexpected or unusual patterns of resource availability.
- **Exception handling:** Presently in workflow systems, exception handling is managed by explicitly representing an alternative path through the business process. In ADEPT, the agents dynamically attempt to renegotiate and re-resource the process task in order to resolve exceptions. This approach allows agents to react to circumstances where the type of corrective action might change depending upon the availability of resources, or how critical the task is within the process as a whole, for example.

The final main differentiator is that workflow management systems tend to operate with a central workflow engine which 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. Similarly, in ADEPT the CORBA platform enables the distribution of agents and business tasks across an enterprise, independent of their location, the underlying operating system, or the hardware being used.

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^{28, 29}. Hall and Shahmehri³⁰ use agent technology to enable both experts (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 are provided in the body, and the completion conditions have a similar role to stop conditions (see figure 4). 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 complementary to agent-based business process management systems such as ADEPT.

A federation-type architecture^{16, 17, 31} 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”¹ 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 ORB²⁴. 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 only been used for the interoperation of purely cooperative agents at the team level of an organisation (e.g. SHADE and PACT¹⁶). 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 work flow³² in business processes. Merz et al.³² 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 while 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 process management system, an organisation must allow intelligent programs from another, possibly competing, organisation to execute on their local machines. Therefore, for mobile agents to be a good implementation choice for agent-based business process management, these security issues must be addressed.

¹. Takeda et al.¹⁷ 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 Ketchpel³¹.

5. Conclusions & Future Work

This paper has described the key components of the ADEPT system and how they were applied to BT's business process of providing a customer quote for network services. 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.
- (ii) **ADEPT as an *implementation* technology:** As well as identifying a methodology, the ADEPT system provides algorithms, interfaces, language definitions, and structures for realising the key concepts. 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.

Implementing the scenario has also identified a number of system and agent issues which require further investigation—these include: the need for richer and more flexible negotiation models, more scalable techniques for sharing information between heterogeneous agents, more elaborate resource management within agents, and more flexible service scheduling algorithms.

Acknowledgements

ADEPT is a collaborative project under the DTI/EPSRC Intelligent Systems Integration Programme (ISIP). The project partners are BT Laboratories, ICI Engineering, Loughborough University of Technology, and Queen Mary and Westfield College. The work described in this paper represents a partial view of the activities of the whole project to which all consortium members have contributed.

References

1. A. Mowshowitz, Social Dimensions of Office Automation, *Advances in Computers* **25** (1996) 335-404.
2. M. J. Wooldridge and N. R. Jennings, Intelligent Agents: Theory and Practice, *The Knowledge Engineering Review* **10** (2) (1995) 115-152.
3. H. J. Müller, Negotiation Principles, in *Foundations of Distributed Artificial Intelligence*, eds. G. M. P. O'Hare and N. R. Jennings (Wiley Interscience, 1996) 211-229.

4. R. G. Smith and R. Davis, Frameworks for cooperation in distributed problem solving *IEEE Trans. on Systems, Man and Cybernetics* **11** (1) (1981) 61-70.
5. N. R. Jennings, Controlling Cooperative Problem Solving in Industrial Multi-Agent Systems using Joint Intentions, *Artificial Intelligence* **75** (2) (1995) 195-240.
6. N. R. Jennings, E. H. Mamdani, I. Laresgoiti, J. Perez and J. Corera, GRATE: A General Framework for Cooperative Problem Solving, *IEE-BCS J. of Intelligent Systems Engineering*, **1** (2) (1992) 102-114.
7. N. R. Jennings, J. Corera, I. Laresgoiti, E. H. Mamdani, F. Perriolat, P. Skarek and L. Z. Varga, Using ARCHON to develop real-world DAI applications for electricity transportation management and particle accelerator control, *IEEE Expert* (1996).
8. N. S. Sridharan, 1986 Workshop on Distributed AI, *AI Magazine*, Fall, (1987) 75-85.
9. J. L. Alty, D. Griffiths, N. R. Jennings, E. H. Mamdani, A. Struthers, and M. E. Wiegand, ADEPT - Advanced Decision Environment for Process Tasks: Overview and Architecture, *Proc. BCS Expert Systems 94 Conference* (Applications Track), Cambridge, UK, 1994, 359-371.
10. J. F. Nash, The Bargaining Problem, *Econometrica* **28** (1950) 155-162.
11. H. Raiffa, *The Art and Science of Negotiation* (Harvard University Press, 1982).
12. J. S. Rosenschein and G. Zlotkin, *Rules of Encounter - Designing Conventions for Automated Negotiation Among Computers*, (MIT Press, 1994).
13. B. Chandrasekaran, Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert System Design, *IEEE Expert* **1** (3) (1983) 23-30.
14. W. J. Clancy, Heuristic Classification, *Artificial Intelligence* **27** (3) (1985) 289-250.
15. J. McDermott, A Taxonomy of Problem Solving Methods in *Automating Knowledge Acquisition for Expert Systems*, ed. S. Marcus (Kluwer, 1988) 225-256.
16. J. M. Tenenbaum, J. C. Weber and T. R. Gruber, Enterprise integration: Lessons from SHADE and PACT in *Enterprise Integration Modelling*, ed. C. J. Petrie, (MIT Press, 1992) 356-369.
17. H. Takeda, K. Iino, and T. Nishida, Agent organisation with multiple ontologies, *International Journal of Cooperative Information Systems*, **4** (4) (1995) 321-337.
18. J. L. Austin, *How to do Things with Words* (Harvard University Press, 1962).
19. J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language* (Cambridge University Press, 1969).
20. M. R. Genesereth and R. E. Fikes (eds.) Knowledge interchange format, version 3 reference manual, Computer Science Department, Stanford University, 1992, Technical Report Logic-91-1. (<http://www-ksl.stanford.edu/knowledge-sharing/papers/>).
21. P. R. Cohen and H. J. Levesque, Communicative actions for artificial agents, *Proc. of the First Int. Conf. on Multi-Agent Systems*, (AAAI Press and MIT Press, 1995) 17-24.
22. T. Finin, D. McKay, R. Fritzson, and R. McEntire, KQML: An Information and Knowledge Exchange Protocol, in *Knowledge Building and Knowledge Sharing*, eds. K. Fuchi and T. Yokoi, (Ohmsha and IOS Press, 1994).
23. T. J. Mowbray and R. Zahavi, *The Essential CORBA Systems Integration Using Distributed Objects* (John Wiley and Object Management Group, 1995).
24. DAIS: A Platform To Build On, ICL Corporate Systems Publications, 1994, Version 2.1.

25. J. C. Giarratano and G. Riley, *Expert Systems: Principles and Programming*, 2nd Edition (PWS Publishers, 1994). (The CLIPS URL is <http://www.jsc.nasa.gov/~clips/CLIPS.html>)
26. Glossary - A Workflow Management Coalition Specification, *Workflow Management Coalition*, November 1994.
27. Barclays Invests in technology to boost customer service, *Technology Strategies*, March 1995.
28. K. Fischer, J. P. Müller, I. Heimig and A-W. Scheer, Intelligent agents in virtual enterprises, *Proc. of the First Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, (1996) 205-223.
29. N. R. Jennings, P. Faratin, M. J. Johnson, P. O'Brien and M. E. Wiegand, Using intelligent agent to manage business processes, *Proc. of the First Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, (1996) 345-360.
30. T. Hall and N. Shahmehri, An intelligent multi-agent architecture for support of process reuse in a workflow management system, *Proc. of the First Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, (1996) 331-343.
31. M. R. Genesereth and S. P. Ketchpel, Software agents, *Communications of the ACM*, **37**(7) 48-53.
32. M. Merz, B. Liberman, K. Müller-Jones and W. Lamersdorf, Interorganisational workflow management with mobile agents in COSM. *Proc. of the First Int. Conf. on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM-96)*, (1996) 405-420.