Implementing a Business Process Management System using ADEPT: A Real-World Case Study

N. R. Jennings[1], P. Faratin[1], T. J. Norman[1], P. O'Brien[2], B. Odgers[2] and J. L. Alty[3]

[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

[3] Dept. of Computer Studies, Loughborough University, Loughborough, Leicestershire, LE11 3TU, UK.
J.L.Alty@lboro.ac.uk

Abstract: This paper describes how ADEPT's agent-based design and implementation philosophy was used to prototype a business process management system for a real-world application. The application illustrated is based on the British Telecom (BT) business process of providing a quote to a customer for installing a network to deliver a specified type of telecommunications service. Particular emphasis is placed upon the techniques developed for specifying services, for allowing agents with heterogeneous information models to interoperate, for allowing rich and flexible inter-agent negotiation to occur, and on the issues related to interfacing agent-based systems and humans. This paper builds upon the companion paper that provides details of the rationale and design of the ADEPT technology deployed in this application.

1 INTRODUCTION

Many advances have been made in recent years within organisations in preparing a culture of dynamic improvement. From the globalisation of trade, Total Quality Management initiatives [4, 5], through regulatory and competitive pressures, and on to ISO 9001-conformant Quality Management Systems, the emphasis has been placed increasingly on the notion of *service*. This embraces both the service that an organisation provides to its customers, and the service that departments within an organisation provide to one another.

To achieve and sustain such dynamic improvement, service-oriented organisations need an infrastructure that supports flexible and robust management of their business activities in a changing world. In particular, the definition and execution of efficient business processes requires considerable effort to be invested in consulting everyone involved and in capturing the requirements and constraints of all the relevant parties. This endeavour involves not only the humans contributing to a process, but also the underlying information systems that facilitate and often have processes embedded within them. To this end, the ADEPT project has developed a business process support infrastructure that is inherently service-oriented, in which both humans and automated tasks can work cooperatively (see the companion paper for the justification and description of ADEPT's conceptual and implementation framework).

In more detail, a business process is composed of a number of primitive functional activities or tasks. In any reasonably complex process, dependencies exist between the tasks and so they have to be executed in a controlled and ordered way. This execution invariably involves the consumption of resources. In most organisations, these resources are grouped into *business units* that control the way in which they are deployed. Within ADEPT, these business units are represented by autonomous software agents. The agents communicate with one another over a network and negotiate over how they can collaborate to manage the overall business process. To be consistent with the service-oriented philosophy, negotiation and collaboration are at the level of the services that agents offer to one another. In this case, a service is a packaging of tasks and other (sub-)services that allows an agent to offer or to receive from another agent some functional operation. A service can be re-used as a component of another service and agents can take the role of provider (server) or customer (client) for services.

Against this background, the purpose of this paper is to demonstrate how the ADEPT concepts and implementation can be used to build a business process management system for a real-world application. For our particular purposes, the exemplar business process is from BT (British Telecom) and it relates to providing a quote for installing a network at a customer's premises. The contributions of this paper are as follows: (i) it demonstrates how a real-world application can be conceived of as a multi-agent system; (ii) it highlights many of the problems that need to be addressed when building a multi-agent system for real-world applications; and (iii) it shows the potential advantages and disadvantages of using multi-agent systems as a software solution technology.

The remainder of the paper is structured in the following manner. Section 2 describes the provide customer quote business process in detail. Section 3 indicates how this business process was divided into agents and agencies and how services were assigned within this structure. Section 4 indicates how the design was realised using ADEPT as a solution technology. Here particular emphasis is given to the negotiation involved in provisioning services and on the techniques for sharing information between agents with heterogeneous information models. Section 5 addresses the issues involved in having both human and artificial agents in a business process management system. Finally, section 6 reflects upon our experiences in building this application and offers some general insights into the problems of building real-world multi-agent systems.

2 BT'S PROVIDE CUSTOMER QUOTE BUSINESS PROCESS

The scenario described in this paper is based on BT's business process of providing a quotation for designing a network to provide particular services to a customer (figure 1). The scenario has been simplified for the purposes of explanation and demonstration. The actual 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 and there are a number of concurrent activities that require coordination.

The overall process receives a 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. The process involves up to six types of business unit: the sales department that is the customer's point of contact; the customer service division that handles the basic administration of the process; the legal department that checks the validity of customers' requests; the design division that determines how the service network can be realised; the surveyor department that visits customers' premises; and the provider of an out-sourced service for vetting customers.

In more detail, the process is initiated by a customer contacting the sales department. The customer's details are captured and whilst the customer is being vetted (in terms of verifying their identity) their requirements are elicted. If the customer fails the vetting procedure, then the quote service terminates. Assuming the customer is satisfactory, their 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, however, the process is more complex. The customer service division 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 service 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 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; 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 customer service division determines whether the customer site(s) should be surveyed. On completion of the network design and network costing, a final costing is produced. Finally, the customer is informed of the service quote and the business process terminates.

In terms of the typical characteristics of corporate-wide business processes that were outlined in section 2.1 of the companion paper, the provide customer quote process has the following traits:

- The resources available to carry out the business processes' activities vary. For example, the numbers of designers, surveyors and lawyers that can work on the process depends upon staff availability at any given time. Similarly, the number of customers that can be checked depends upon the current capacity of the vetting agencies. Moreover, for a given customer, it is not possible to predict when they first enter the system which path through the business process will be enacted.

- The process involves approximately equal amounts of human and automated tasks. Tasks such as checking the legality of services, surveying a customer's premises, and designing the network are performed by humans, whereas tasks such as costing portfolio services and delivering the final quote are automated.

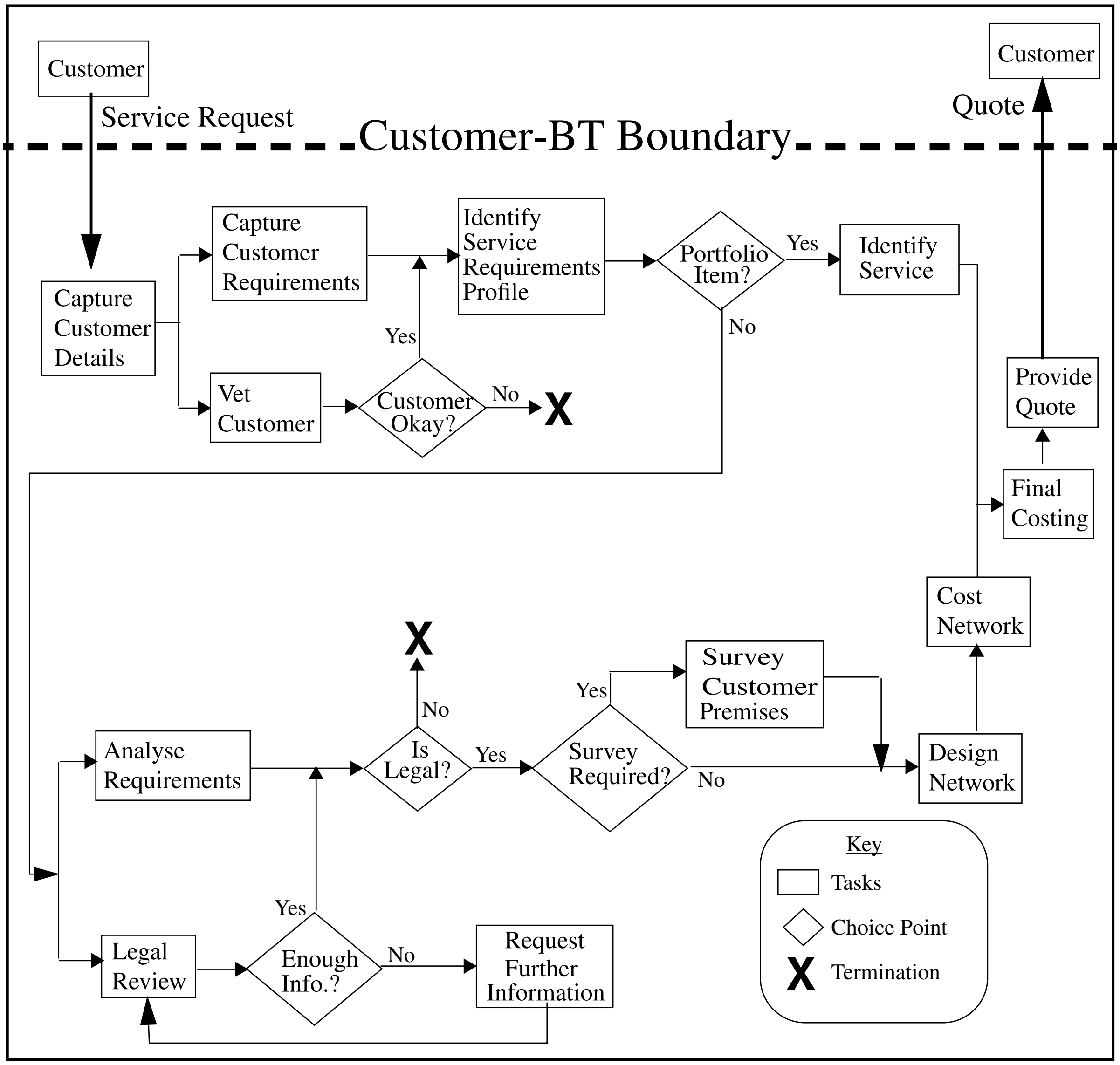


Figure 1: The Provide Customer Quote Business Process

- Multiple organisations are involved in the process. BT is responsible for the majority of the activities, but the task of vetting customers is outsourced to external companies that bid for contracts on an as-needed basis.

- The process is physically distributed across different resource units that are situated at various BT sites. Also the vetting companies can be located anywhere in cyberspace.

- The resources involved in the process are controlled by their respective business units in an autonomous way. Thus, the design department determines the number of designers that are available for this process (bearing in mind that this is not the only activity in which the design department is engaged). Similar observations are true for the surveyor and legal departments.

- There are many concurrent problem solving activities—both within a given process instance and across process instances.

- The business process manager needs to have a view of the entire process so the efficiency of the activity can be determined. Thus, information is needed about the process instances that take significantly longer than average, about where bottlenecks occur, and so on.

Given these observations, it is clear that the provide customer quote process exhibits most of the typical characteristics of corporate-wide business processes in large organisations. For this reason, it represents a suitable and realistic context in which to test the operation of the ADEPT system.

3 DESIGNING THE AGENT SYSTEM

This section describes how the customer quote business process was mapped into a multi-agent system. The main steps in this endeavour are determining which agents should be present and how they should be organised into agencies (section 3.1), specifying the services the agents should provide to one another (section 3.2), and setting the mode of provision for the identified services (section 3.3).

3.1 Mapping the business process into agents and agencies

Since the ADEPT solution has to operate within the present organisational structure, our first step was to identify the business units involved in the customer quote activity. As indicated previously (section 2), there are five business units within BT involved in this activity: the sales department, the customer handling department, the design department, the legal department, and the survey customer site department. Given the fact that these business units represent autonomous problem solving entities with their own set of resources, the obvious design choice is to make each of them an agent. However this position is complicated by the fact that customer handling is dealt with at two physical sites—London and Edinburgh. Whilst it would obviously be possible to view the two sites as one logical entity, we decided upon two separate agents since there is comparatively little coordination between the sites and there are no control relationships between the sites. The final type of entity involved in the business process is the customer vetting organisations. As already explained, vetting is out-sourced. As there are multiple potential providers of the customer checking activity, it seems natural to represent each of them as separate agents since they are, in effect, direct competitors. Thus, the implemented system contains the following agents (figure 2): the BT Sales (BTS) agent; the Customer Handling in London (CHL) agent; the Customer Handling in Edinburgh (CHE) agent; the Network Design Department (NDD) agent; the Telecommunications Legal Service (TLS) agent; the Surveyor Team (ST) agent; and the three external agents—Customer Information Centre (CIC) agent, Financial Services Enterprises (FSE) agent, and Customer Check Ltd. (CCL) agent—that provide the customer vetting.

The next step down from the business unit is the individuals working within the units (e.g. the surveyors that visit premises, the lawyers that check the legality of proposed services, etc.). These entities could validly be represented as agents in their own right. However for reasons of simplicity we decided against this design choice in this particular application. Instead, we chose to represent these entities as tasks. This enabled resource allocation (assigning individuals to work items) to be performed by a business unit wide scheduling system. For example, the surveying department already has a system that schedules the work of the surveyors so as to optimise factors like the distances between jobs and the amount of overtime required. Therefore in this case, the agents represent the interests of the pool of resources and the resource utilisation information used during service provisioning originates from the work scheduling system. We found this approach to be more efficient than having each agent represent the interests of an individual resource.

Having identified the set of agents, the next step is to determine the organisational relationship between them. In the ADEPT context, this equates to determining which agents are peers of one another and which are sub-agencies. In the provide customer quote case, this is, in general, a fairly easy choice since most agents represent distinct, autonomous departments that have no control authority over one another. The exception to this is the survey department that is part of the design department. Thus, it was decided that all the agents should be peers apart from the ST agent that would be a sub-agency of the NDD agent. In the latter case, having ST as a sub-agency is a natural choice since only NDD needs to interact with it. Moreover, the ST agent is an integral part of the NDD's functioning since the design service cannot be performed without having a surveying capability. This means the two agents have a close and cooperative working relationship which is exactly that captured by the sub-agency concept.

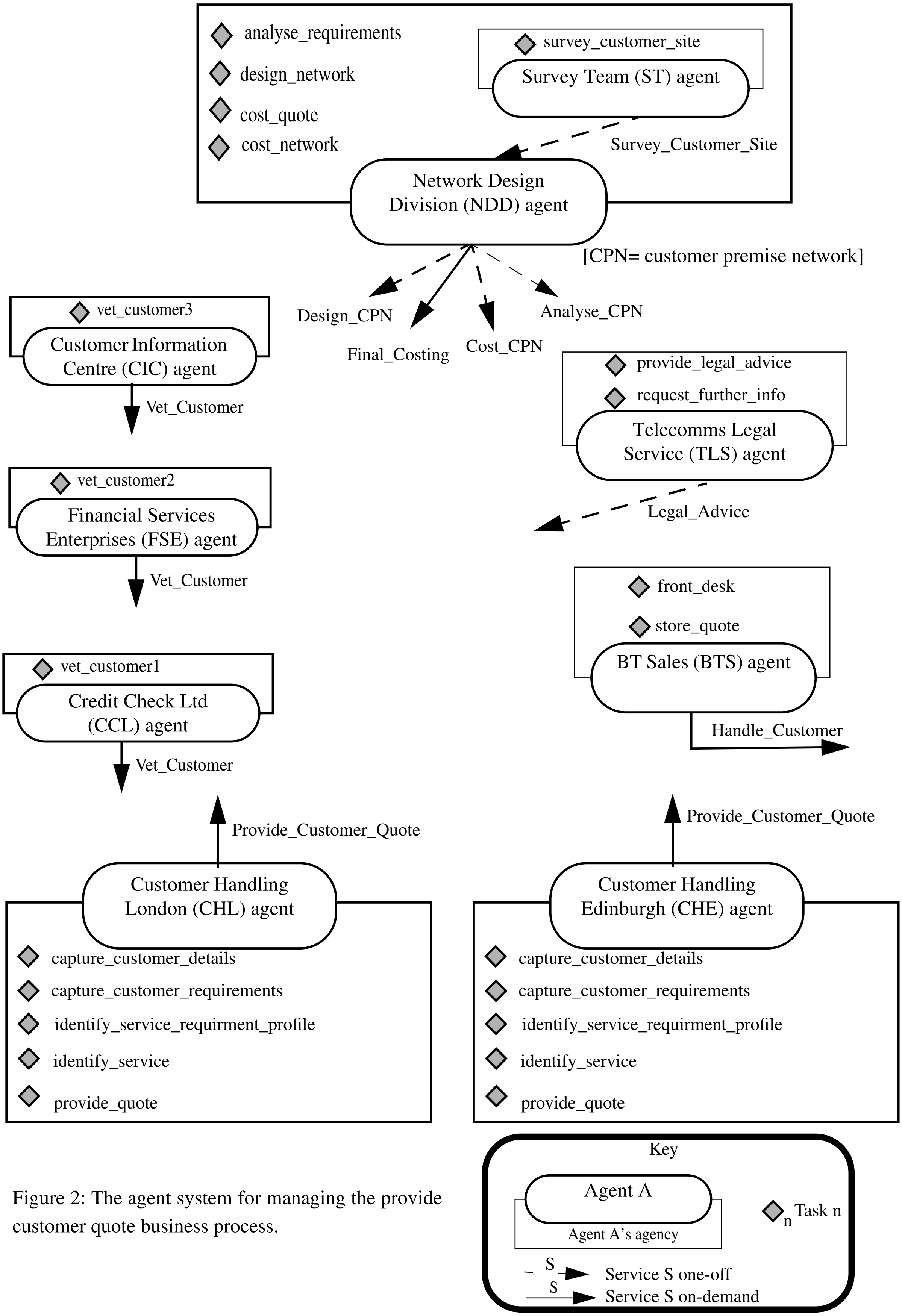


Figure 2: The agent system for managing the provide customer quote business process.

3.2 Assigning Services to Agents

Having identified the agents involved in the business process, the next step is to determine the services they will provide to one another. In most cases, this question can be answered by considering the roles of the various agents and by applying the software engineering principle of minimal coupling between problem solving groups [14][1]. Having said this, however, it is important to note that the composition of services offered by an agent is determined by the requirements of *all* the business processes within an enterprise, not just one. Hence some services are necessarily defined at a level of granularity that does not appear as natural for one application, but which makes sense when the totality of the business processes are considered.

- The BTS agent handles customer service enquiries on behalf of BT. This service (`Handle_Customer`) encapsulates the entire quote process and represents the customer's sole point of contact. To perform this role, BTS requires one of the customer handling departments to provide a quote.

- The vet customer agents (CIC, FSE and CCL) provide a customer checking service (`Vet_Customer`) to the customer handling agents (CHE and CHL).

- The ST agent provides the service of surveying a customer's premises to the NDD agent (`Survey_Customer_Site`).

- The customer handling agents provide the (identical) service of providing quotes to the BTS agent (`Provide_Customer_Quote`). To provide this service, the customer handling agents need to be able to call upon a vetting service, a legal advice service, and a costing and design service.

- The TLS agent provides advice on whether a particular customer's request is legal in the country where the network is to be situated. This service (`Legal_Advice`) represents an important choice point in the process. If the proposed request is illegal then the quote process must terminate and the customer must be informed. Therefore this service is provided to the customer handling agents (rather than the NDD agent directly) since they are responsible for liaising with the customers.

- The NDD agent provides the services of analysing the requirements of a bespoke customer premise network (CPN) (`Analyse_CPN`), of designing a CPN (`Design_CPN`), and of costing the network (`Final_Costing`, `Cost_CPN`) to the customer handling agents. These services are used in business processes other than that for providing a customer quote and so they are represented separately even though they are always invoked together in this particular context.

Having specified the agents' services, the next step is to determine in which agents the constituent tasks (see figure 1) should be located so that the services are realised in the most effective fashion. To minimise the coupling between agents, tasks are managed by the agent that provides the service of which they are part unless an explicit statement is made to the contrary.

- The `Handle_Customer` service requires the BTS agent to provide a point of contact for the customer (the `front_desk` task) and the ability to store information about a customer's request while the quote process is being enacted (the `store_quote` task).

- The `Vet_Customer` service is composed of a single task (`vet_customer`). Since the exact means by which this service is realised will vary between the agents, the specific tasks also vary (hence the 1, 2 and 3 subscripts in figure 2).

- The `Legal_Advice` service is composed of two basic tasks: providing the advice (`provide_legal_advice`) and requesting additional information when it is needed (`request_further_info`).

- The `Survey_Customer_Site` service is composed of a single task (`survey_customer_site`).

---

1. Indeed subsequent work has expanded upon this idea to produce a methodology for designing and building agent-oriented software [15]

- The Provide_Customer_Quote service involves the following tasks: capturing details about the customer (capture_customer_details); determining the customer's requirements (capture_customer_requirements); identifying the desired service's requirements profile (identify_service_requirement_profile); identifying the particular service if it is a portfolio item (identify_service); and providing the final quote to the BTS agent (provide_quote).

- The Analyse_CPN service involves analysing the customer's requirements in more detail (analyse_requirements). The Design_CPN service is where the bespoke network is designed (through the design_network task). The Cost_CPN service uses the cost_network task to calculate the actual cost of the customer design. The Final_Costing service calculates the final quote for the customer that might include additional value added services (e.g. maintenance contract options) and ensures the quotation is appropriately authorised (both for bespoke and portfolio items) through the cost_quote task.

3.3 Determining Modes of Service Provision

Having determined the agents' services, the final design choice relates to the way these services will be provisioned. As indicated in section 3.1.2 of the companion paper, services can be provisioned in two ways: (i) *on demand*: one agreement covers multiple invocations of a particular service; and (ii) *one-off*: a single service instance is provisioned at a time. The former is used for services that have a high volume of invocation. This means they are usually on the main path of the business process. The latter is used for services that are invoked less frequently. The rationale for this is that negotiation consumes both time and resources and hence for services that are used frequently it makes sense for the cost of the negotiation to be amortised over numerous services. However the main advantage of negotiating afresh for each service instance is that the agreement can be tailored more accurately to the prevailing circumstances. Hence for services that are invoked infrequently, the increased resource overheads tend to outweigh the timeliness of the agreements.

In the provide customer quote process, it was decided that the following services should be made on-demand since they are required for each and every invocation of the business process: Handle_Customer, Provide_Customer_Quote, Vet_Customer and Final_Costing. The remaining services are provisioned as and when they are needed: Legal_Advice, Survey_Customer_Site, Design_CPN, Cost_CPN and Analyse_CPN.

4 IMPLEMENTING THE AGENT SYSTEM WITHIN ADEPT

This section describes the major steps involved in implementing the provide customer quote business process using ADEPT as a solution technology. Section 4.1 discusses the issues associated with specifying services in ADEPT's service description language; section 4.2 shows how the ADEPT approach to information sharing is used in practice; section 4.3 illustrates service provisioning in a range of negotiation contexts; and section 4.4 demonstrates the service management function in operation.

4.1 Specifying Services

Services are specified using the Service Description Language (SDL) introduced in the companion paper. Each service 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 that is to be used by the service and the outputs specify the information provided to the client on completion of the service. The guard is a boolean expression that is evaluated when the service is first invoked. If this expression evaluates to false, the service fails without the body of the service being processed. The body of a service description loosely specifies how the service is to be executed. It consists of the restrictions on the order in which component services are to be executed, the conditions under which the service will be considered successfully executed, and how information flows between these component services.

A service description represents the server agent's knowledge of how to provide a service. However, a potential client agent does not have this knowledge since the service's realisation is encapsulated by the provider. Therefore, services are advertised as *service prototypes*. A service prototype provides a client agent with the necessary information with which to manage the execution of a service provided by a server agent, given that there exists an appropriate agreement between them. This information consists of the name of the service, its inputs and outputs. An agent also represents the tasks it is responsible for managing in a similar way, providing the name of the task, its inputs and outputs. Sample service and task prototypes for the CHE agent are shown in figure 3. The service prototype refers to the service `Vet_Customer`, and indicates there is a single input to this service, the details of the customer concerned, and a single output, the verdict of the `Vet_Customer` service provider. Note that unlike a service description, the inputs to a service as specified in a service prototype only refer to the input as being mandatory (`man`) or optional (`opt`). Inputs that are provided by the server itself are not specified. The task prototype refers to the task `capture_customer_details`. There is one mandatory input to this task and a single output: the telephone number of the customer and their full details respectively. The task is therefore to contact the customer by phone and determine details such as their postal address and fax number.

```
(service_prototype
  name    Vet_Customer
  inputs  ( BT_Customer_Details customer_details man )
  outputs( BT_Customer_Credit_Verdict verdict ) )

(task_prototype
  name    capture_customer_details
  inputs  ( BT_Customer_Details telephone man )
  outputs( BT_Customer_Details custDetails ) )
```

Figure 3: Sample service and task prototypes for the CHE agent

The inputs and outputs of service prototypes are always defined in terms of the client agent's information model. This may be different from the server's information model, in which the server agent's service description is defined. For example, the service description of the `Vet_Customer` service (figure 4) that may be provided by the CCL agent is expressed in terms of the internal information model of that organisation. It is important to note two features of a client's service prototype and the associated server's service description. First, the name of the service must be identical; service names are common to all agents whatever information model is used. Second, each information object is prefixed with the name of the information model in which it is defined. The information objects used to represent the inputs and outputs of CHE's service prototype are prefixed by "`BT`", that represents the British Telecom information model, and CCL's service description uses information objects prefixed by "`CCL`", that represents that organisation's information model. (Refer to section 4.2 for information on how information is shared between these agents during service execution.)

```
(service
  name    Vet_Customer
  inputs  ( CCL_Customer customer_details cli man )
  outputs( CCL_Decision verdict )
  guard   ("")
  body (
          sequence {
                  check_CCL
                          ( details = service::customer_details
                            service::verdict = limit )
                  } ->
                  ( check_CCL ) ) )
```

Figure 4: SDL description of the `Vet_Customer` service provided by the CCL agent

CCL's `Vet_Customer` service description is rather simple, consisting of a single task to be performed. The body of this service description contains a single sequence block with the completion condition that the task `check_CCL` is successfully completed. The content of this block is a call to that task, where the task's input (called `details`) is the `CCL_Customer` information object that is input to the `Vet_Customer` service, and its output is the `CCL_Decision` information object called `verdict` that is the service's final output. To illustrate the use of SDL more fully, the service description for the NDD agent's `Design_CPN` service is shown in figure 5.

```
(service
  name   Design_CPN
  inputs ( BT_Customer_Details custDetails cli man
           BT_Survey_Required survey cli man
           BT_CR_Profile crProfile cli man )
  outputs( BT_Network_Design networkDesign )
  guard  ("")
  body (
      sequence {
         sequence:doSurvey {
                cond:needSurvey("True = service::survey.survey_reqd"),
                Survey_Customer_Site
                ( customerDetails = service::custDetails
                  crProfile = service::crProfile
                  z = cpeSpecification )
                } ->
                ( or (not needSurvey) Survey_Customer_Site ),
         design_network
         ( customerDetails = service::custDetails
           analysis_details = service::crProfile
           cpe = z
           service::networkDesign = design )
              } ->
         ( and doSurvey design_network ) )
```

Figure 5: SDL description of the `Design_CPN` service provided by the NDD agent

The three inputs to this service are all mandatory and are to be provided by the client, and so each of these will be mentioned in the client's service prototype. These inputs are the details of the customer, an information object that indicates whether a survey is required of the proposed network site, and the profile of the customer's requirements. The service's output is a single information object representing the completed network design. The service has no guard and so the body is always executed. The body consists of two sequence blocks, and refers to a task (`design_network`) and a service (`Survey_Customer_Site`). The main block is considered to have been completed, and hence the service is completed, if both of the symbols "`doSurvey`" and "`design_network`" evaluate to true. The `doSurvey` symbol evaluates to true when the sequence sub-block with the same name is successfully completed, and the symbol `design_network` evaluates to true when the task with the same name is successfully completed. The `doSurvey` sub-block and the `design_network` task are specified to be executed in sequence and so the `doSurvey` sub-block is always executed first. This sub-block is completed if the condition `(or (not needSurvey) Survey_Customer_Site)` evaluates to true. This is the case if either `needSurvey` evaluates to false or `Survey_Customer_Site` evaluates to true. This is a sequential sub-block, and so each element in the execution list is executed in strict order. The first element is a conditional test, that determines the truth condition of the `survey_reqd` slot within the `survey` input to this service. This information object is of type `BT_Survey_Required`, defined as follows:

```
(class BT_Survey_Required
  (Types_String cr_profile)
  (Types_Boolean survey_reqd) )
```

This information object associates a string that identifies the customer requirements profile and a boolean indicating whether or not a survey is required for this order. (Note that a single customer may place multiple orders at the same time, and so it is essential to distinguish between them by indicating which requirements profile this refers to.) The `survey_reqd` boolean is referred to in the service description by using the keyword `service`, that indicates that this information object is an input or output of the service being described, followed by a double colon "`::`", the variable denoting the information object `survey`, dot "`.`", and the slot `survey_reqd`. Thus, this conditional tests the truth condition of this boolean. If it is false, the conditional evaluates to false, the completion condition of the `doSurvey` sub-block becomes `(or true Survey_Customer_Site)` (where the completion condition of `Survey_Customer_Site` is unknown at this time), and hence the block has been successfully completed. If a survey is required, the completion statement of this sub-block becomes (or false Survey_Customer_Site), and hence the completion condition of Survey_Customer_Site must be determined to evaluate the completion condition of this sub-block. Thus, if a survey is required, the Survey_Customer_Site service is executed.

Assuming that there exists an agreement with an agent that is able to provide the Survey_Customer_Site service, the agent proceeds to execute the service. The inputs to the service are the customer details and customer requirement profile that are provided as input to the Design_CPN service, and there is a single output that is bound to the variable “z”. If this service is successfully completed, its completion condition becomes true, and hence the completion condition of the doSurvey sub-block becomes true. The execution of this service then proceeds to the task design_network. The task prototype for this task is given below:

```
(task_prototype
  name    design_network
  inputs  ( BT_Customer_Details customerDetails man
            BT_CR_Profile analysis_details man
            BT_CPE_Specification cpe opt)
  outputs ( BT_Network_Design design ) )
```

Note that the variable z only has a value if the Survey_Customer_Site service has been executed. This is reflected in the task prototype, where the third input to the task is specified as optional. The output of this task is bound to the variable service::networkDesign; i.e. the output of the Design_CPN service. If both the design_network task and the doSurvey sub-block are successfully completed (i.e. both symbols design_network and doSurvey evaluate to true), then the service Design_CPN is successfully completed, as specified in the completion condition of the service.

This example illustrates the power of the SDL. With this language, it is possible to specify the same service in a number of ways. For example, the Design_CPN service description could equally be expressed without the doSurvey sub-block by combining the completion condition of this sub-block with the completion condition of the task design_network. However, through experience, we have found that the use of sub-blocks gives structure to the service description, simplifies the completion conditions of each block, and aids both the understanding and debugging of service descriptions when building an application. In short, this is simply good software design.

4.2 Information Sharing

Information that is shared by two or more agents must have a common interpretation. However, it was seen in the previous section that agents within the provide customer quote application do not all share a common model of information. For example, agent CHE uses the BT information model, and CCL uses an information model that is particular to the organisation it represents. The inputs and outputs to services are specified in the agent’s local information model within their service descriptions. Therefore, the question that must be addressed is how agents with differing information models communicate in the provisioning and execution of services. The approach adopted in the ADEPT system is to use a common information model in which all information is referred to and passed during inter-agent communication (see section 3.2 of the companion paper for more details). The common information model used in this application is the BT information model[2]. To avoid confusion, information objects within a particular information model are prefixed by the name of that model; e.g. information objects in the BT information model are prefixed with the string “BT_”.

To illustrate information sharing in ADEPT, consider the provision and execution of a Vet_Customer service. This service is available from three external agents: FSE, CCL and CIC, each of which uses a different internal model of information (since they represent three different companies). Both agents CHL (customer handling in London) and CHE (customer handling in Edinburgh) require services of this kind when providing a quote. However the information they require is different. CHE requires a BT_Customer_Credit_Verdict information object. This associates a unique string that identifies the customer concerned, with a boolean that indicates whether or not the customer’s credit is good.

2. Note this model is also used as an internal information model by several of the agents.

```
(class BT_Customer_Credit_Verdict
   (Types_String customer_details)
   (Types_Boolean customer_credit_acceptable) )
```

CHL requires a `BT_Customer_Credit_Rating` information object. This associates a unique string that identifies the customer concerned, with a float representing the estimated credit limit of the customer and an integer representing the confidence of this estimate.

```
(class BT_Customer_Credit_Rating
   (Types_String customer_details)
   (Types_Float customer_credit_limit)
   (Types_Integer confidence) )
```

Furthermore, the information that is available from CHE and CHL as input to a `Vet_Customer` service is the details of the customer to be vetted; i.e. a `BT_Customer_Details` information object. Therefore, for FSE, CCL and CIC to be able to provide a `Vet_Customer` service to CHE and CHL, they must make use of the information available (i.e. `BT_Customer_Details`) when they act to produce the outputs required (i.e. `BT_Customer_Credit_Verdict` or `BT_Customer_Credit_Rating`).

By using its acquaintance models, CHL will be able to identify that FSE, CIC and CCL are all able to provide the `Vet_Customer` service. CHL will then propose a service level agreement (SLA) to each of these potential service providers (see section 4.3). In addition to specifying the service's price, time scale, etc., the SLA indicates the inputs that may be provided and the outputs that are expected by the client. On receipt of a proposal, it is the job of the agent to ensure that the inputs are acceptable and the outputs may be provided as specified. In this case, both FSE and CIC are able to translate the information that is provided by its tasks (`FSE_Verdict` and `CIC_Rating` respectively) into the `BT_Customer_Credit_Rating` information object required by CHL, but CCL cannot[3] (see figure 6). The information that is available from CCL's tasks is `CCL_Decision`, which may only be translated into an instance of `BT_Customer_Credit_Verdict`. Therefore, unless CHL will accept a `BT_Customer_Credit_Verdict` information object instead, CCL cannot provide this service to CHL, and negotiation will terminate.

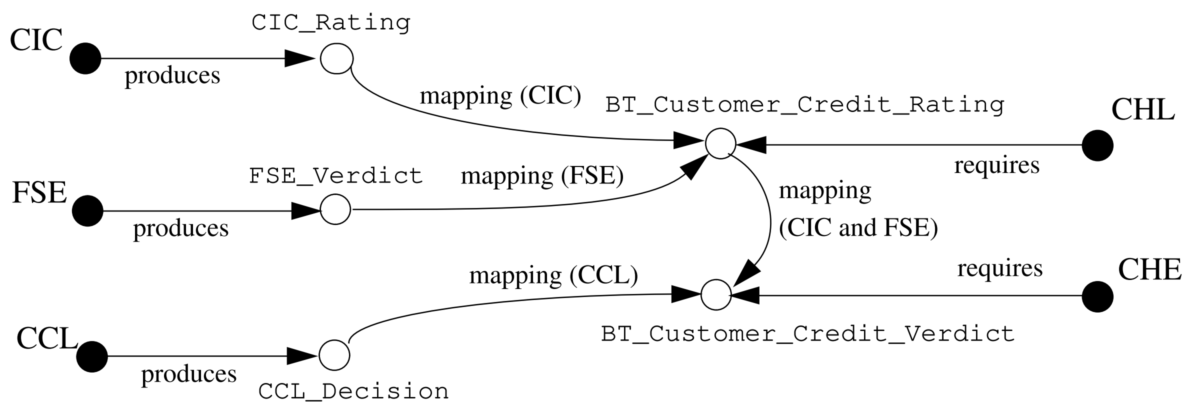


Figure 6: Outputs and translations from the various `Vet_Customer` services.

During service execution, information that is provided by the client must be transformed into the common information model by the client, transmitted to the server, and then transformed by the server into its local information model. Information that is then to be provided to the client as output must be transformed into the common information model by the server, transmitted to the client, and then transformed by the client into its local information model. In this example, the client (CHL or CHE) uses the common information model

3. The ability to translate either means that a stored information model mapping exists or that the search engine is able to find an appropriate ontology description (figure 6 of the companion paper). If neither of these options are available, then the agents will not be able to attain sufficient commonality to interact purposefully.

internally, and so information translation is performed by server agents only (FSE, CIC or CCL). Suppose that CIC has an agreement to provide a number of instances of a `Vet_Customer` service to CHL and that CHL has invoked this agreement. The agreement specifies that CHL must pass CIC the information concerning the customer to be vetted. After vetting this customer, CIC must communicate the results, that are represented by an instance of the information object class `CIC_Rating`, to CHL. CIC has a method (a stored information model mapping) of transforming an information object of type `CIC_Rating` to the required output, `BT_Customer_Credit_Rating`. This translation is performed by the communication module, and the output communicated to CHL. Suppose that FSE has an agreement with CHE for the same service, and wishes to communicate the results of executing this service to CHE. FSE has no direct mapping from `FSE_Verdict` to `BT_Customer_Credit_Verdict` (see figure 6), and so it searches for a method of performing the required translation. In this case, a valid mapping can be found by concatenating the mapping between `FSE_Verdict` and `BT_Customer_Credit_Rating`, and the mapping between `BT_Customer_Credit_Rating` and `BT_Customer_Credit_Verdict`. These methods may then be combined to create a single direct mapping from `FSE_Verdict` to `BT_Customer_Credit_Verdict` for future use (see section 3.2 of the companion paper for a discussion of this matter).

4.3 Service Provisioning

This first part of this section illustrates the process of providing agents with the necessary domain information on which to base their negotiation behaviour (section 4.3.1). It then goes on to describe the operation of the agents' negotiation reasoning model in two service provisioning scenarios (section 4.3.2).

4.3.1 Specifying the Domain-Dependent Negotiation Information

For service provisioning, the first task of an ADEPT system designer is to specify the structure of the negotiation object (i.e. define the SLA template—point ii of section 2.4 in the companion paper). This template should include all those issues that represent important dimensions/attributes of a service that need to be settled during the service provisioning phase. In general, the set of negotiation issues may vary between applications, but some attributes (e.g. price, duration, penalty and volume) are usually present. In the BT application, the set of issues is fixed and common to all agreements—see figure 7 for a list of all the attributes used in this application. (The precise meaning of each slot is discussed in section 3.3.2 of the companion paper.) It is important to note that some of these issues are quantitative in nature (e.g. price, volume) and so the negotiation space is continuous, whereas other issues are qualitative in nature (e.g. outputs required, mode of provision) in which case the negotiation space is discrete (e.g. `x` is produced/not produced by the service, service is provisioned as one-off or on-demand).

client: CHL	Issue Ranges	Weight
DELIVERY_TYPE:	on-demand	0.1
DURATION: (minutes)	{60,120}	0.1
START_TIME:	{6:00,9.00}	0.1
END_TIME:	{14:00,18:00}	0.1
VOLUME:	{30, 100}	0.15
PRICE: (per costing)	{1,20}	0.1
QUALITY:	{0.5,1.0}	0.15
PENALTY:	{3,10}	0.1
CLIENT_INFO:	BT_Customer_Details	0.05
REPORTING_POLICY:	BT_Customer_Credit_Rating	0.05

server: CIC	Issue Ranges	Weight
DELIVERY_TYPE:	on-demand	0.1
DURATION: (minutes)	{110,200}	0.05
START_TIME:	{8:30,9:30}	0.1
END_TIME:	{14.00,18:00}	0.1
VOLUME:	{25,40}	0.1
PRICE: (per costing)	{3,5}	0.3
QUALITY:	{0.1,0.6}	0.05
PENALTY:	{1,3}	0.1
CLIENT_INFO:	BT_Customer_Details	0.05
REPORTING_POLICY:	BT_Customer_Credit_Rating	0.05

Figure 7: Reservation values for a `Vet_Customer` service negotiation: client (left) and server (right)

Once the negotiation issues have been identified, the designer must specify, in the agent's self model, the *reservation values* and the relative importance (or weights) for each issue (point iii of section 2.4 of the companion paper). Reservation values represent the limits of acceptable values for that service issue. For example, the highest price the agent is willing to pay for a service or the shortest duration for completing the service. The weights indicate the relative importance of each issue. For example, in some cases price might be the dominant issue, and in others quality may be more important.

To illustrate these points, consider the negotiation between the CHL and the CIC agents for the `Vet_Customer` service (figure 7). For each quantitative issue, the minimum and maximum acceptable values are specified. Thus, CHL is willing to pay between £1 and £20 for the `Vet_Customer` service, while CIC will only perform the service if it receives at least £3. Negotiations over qualitative values differ to the extent that there is less scope for compromise. In fact, the qualitative issues in this application domain have a single discrete value. Thus the fact that the server needs the client to provide information on customer details is not something that CHL can negotiate over. If it is not provided, then CIC simply cannot perform the service. The same is true if a given server can only provide one-off scheduling of services and does not support on-demand provisioning; again it is an innate property of the agent and it is not something that can be argued over during the negotiation. Thus negotiation over qualitative issues is more a process of satisfying and delivering on constraints.

On some issues, the client and the server may have mutual interests. In the above example, both CHL and CIC have mutual interests with respect to the delivery type and the type of information that each agent will provide for the service. However for most issues, clients and servers have opposing interests. For example, CHL would prefer CIC to start the service earlier in the day, take a shorter time for each vetting instance, process more customers for a lower payment and incur a higher penalty for breaking the contract. Therefore, in the agreement space, clients and servers represent opposing forces on most issues and it is only through the process of negotiation that mutually acceptable positions can be reached[4].

The client and the server also differ in the importance they attach to each of the negotiation issues. In the above example, CHL assigns the highest importance to the volume of service invocations that can be handled and the quality to which the vetting is carried out. CIC, on the other hand, assigns the most importance to the price it pays for the service.

4.3.2 An Execution Trace of ADEPT's Negotiation Model

To illustrate the operation of ADEPT's negotiation model, this sub-section describes two exemplar service provisioning episodes in detail (see section 3.3 of the companion paper for more details of the model itself). The first involves the negotiation between the customer vetting agents and the customer handling agents over the `Vet_Customer` service. The second involves the negotiation between a customer handling agent and the NDD agent for the `Design_CPN` service. The former can be characterised as competitive, multi-lateral, inter-organisational negotiation, whereas the latter is a cooperative, bi-lateral, intra-organisational process. In order to focus attention on the key aspects of the reasoning model, we consider a subset of the negotiation issues. Moreover, in the first scenario we concentrate on the decisions involved in setting the logistics of negotiation and the strategy modification reasoning, while in the second scenario we focus on the tactical and evaluatory aspects of the model.

4.3.2.1 Negotiating for the `Vet_Customer` Service

Assume the negotiation set for the `Vet_Customer` service is composed of three issues: the *price* paid per invocation, the *volume* of invocations between the contract's start- and end- times, and the *penalty* for breaking the contract. For each issue, both the clients (CHL and CHE) and the servers (CIC, FSE, CCL) have preferences over the reservation values and have rankings of the relative importance of the various issues (figure 8). Compared to CHE, CHL is willing to pay more for the service, requires a larger number of service invocations, and demands a higher penalty for failure. CHL is indifferent to the importance of the price and penalty issues, but holds the volume of the service as the most important issue. CHE, on the other hand, is indifferent between the volume and penalty of a service, but considers price to be the most important issue in the negotiation set. The least expensive `Vet_Customer` service is provided by CIC that holds price as the most important negotiation issue. However, it offers a lower volume of invocations than FSE and CCL and it prefers the lowest penalty for contract violation. FSE, on the other hand, typically requires a higher payment for the service, can provide more instances of the serv-

---

4. There must be an overlap in the reservation values for each issue for an agreement to be possible. However since the reservation values are private to the agents, this information may only emerge during the negotiation process.

ice, is prepared to accept higher penalty rates, and is indifferent between the negotiation issues. The largest capacity agent, CCL, demands the highest prices for its service and is prepared to pay higher penalties. It considers the penalty to be the least important issue and it attaches equal weight to price and volume.

	CHL	CHE	CIC	FSE	CCL
[$price_{min}$, $price_{max}$]: $weight_{price}$	[1,20]: 0.05	[1,15]: 0.5	[3,5]: 0.8	[1,10]: 0.33	[10,40]: 0.4
[$volume_{min}$, $volume_{max}$]: $weight_{volume}$	[30,100]: 0.9	[20,80]: 0.25	[25,40]: 0.1	[30,60]: 0.33	[25,110]: 0.4
[$penalty_{min}$, $penalty_{max}$]: $weight_{penalty}$	[3,10]: 0.05	[1,5]: 0.25	[1,3]: 0.1	[1,5]: 0.33	[1,10]: 0.2

Figure 8: Reservation values and relative importance weightings for the `Vet_Customer` negotiation

To complete the negotiation context, we need to know about the agents' negotiation deadlines and about any previous interactions. In terms of deadlines, CIC has a fairly short deadline, FSE, CHL and CHE have medium deadlines and CCL is under no significant time pressure. Further assume that CHL has had no previous interactions with the servers, but that CHE has and that it has developed the preference ordering FSE > CCL > CIC. Finally, assume the negotiations start at different times; CHL first, followed by CHE.

Agent		***Time***				
From	*To*	t_1	t_2	t_3	t_4	t_5
CHL	*CIC*	[0.0,0.0,1.0]		[0.2,0.0,0.8]		accept
	FSE	[0.0,0.0,1.0]		[0.4,0.0,0.6]		reject
	CCL	[0.0,0.0,1.0]				
CHE	*CIC*					
	FSE		[0.6,0.0,0.4]		[0.5,0.0,0.5]	
	CCL					
CIC	*CHL*		[0.8,0.0,0.2]		[0.9,0.0,0.1]	
	CHE					
FSE	*CHL*		[0.6,0.0,0.4]		[0.3,0.4,0.3]	
	CHE			[0.3,0.6,0.1]		accept
CCL	*CHL*		reject			
	CHE					

Figure 9. Strategic changes during the course of a `Vet_Customer` negotiation.
3-tuples represent the respective agent's strategy; indicating the respective weights of the time-, and behaviour- dependent tactics.

Given this context, we will now step through a sample negotiation concentrating predominantly on the logistics and strategy modification aspects of the reasoning model (figure 9). Details of the reasoning associated with generating the specific offers and in evaluating offers are given in the second scenario (section 4.3.2.2). In this case, an agent's strategy is represented as a 3-tuple [x_1, x_2, x_3] where x_1 represents the weight assigned to the time-dependent tactic, x_2 represents the weight attached to the resource-dependent tactic, and x_3 represents the weight attached to the behaviour-dependent tactic.

CHL initiates the negotiation at t_1 by proposing the same SLA to *all* of the servers in parallel since it has no preferences over the servers and has enough time to negotiate (rule 2 of figure 11 of the companion paper). Since all the potential servers are from a different organisation and time and resources are not yet a consideration, CHL's strategy is to base its negotiation stance on the behaviour of its opponent (rule 1 of figure 12 of the companion paper).

After evaluating the proposed SLA, each potential server responds at t_2. In this example, FSE and CIC find the proposed SLA unacceptable and counter-propose an alternative. Since this is a competitive negotiation and there are no previous interactions between the participants, each server sets a strategy that to some degree is based on CHL's behaviour (rule 1 of figure 12 of the companion paper). CIC's, and to a lesser extent FSE's, strategies place a large emphasis on the time remaining since they have a comparatively short time in which to make an agreement (mixture of rules 1 and 3 in figure 12 of the companion paper). CCL rejects CHL's offer since it cannot map between the output of its vetting service (`CCL_Decision`) and the information required by CHL (`BT_Customer_Credit_Rating`) (see section 4.2).

Also assume that CHE starts its negotiation at t_2 by making an offer to FSE as its most preferred server (rule 1 of figure 11 of the companion paper). Given the previous successful interactions between this pair, CHE adopts a strategy that places greater emphasis on the time to reach an agreement than on the behaviour of the opponent (i.e. the degree of trust that exists between the agents means that CHE does not adopt a pure behaviour-based strategy (cf. CHL's strategy for the same negotiation)).

At t_3, CHL receives back the two counter proposals and evaluates them. The one from CIC has the highest utility since it will have conceded the most (it is under the most time pressure). FSE will have conceded, but not as much. Assuming neither of these offers are acceptable, CHL needs to prepare new counter proposals. Before doing this, however, it determines whether it should modify its strategies. Since time is starting to run short, it needs to switch its strategies from being purely behaviour-based to ones that take greater account of time (rule 1 of figure 13 in the companion paper). Since CIC is conceding more rapidly, CHL tries to exploit it to maximise its gain (rule 3 of figure 13 of the companion paper). Therefore it places a greater emphasis on the time remaining, than it does on the behaviour of the opponent. FSE has conceded less than CIC, and so CHL attaches a greater weight to its time dependent tactic.

Also at time t_3, FSE evaluates CHE's offer. It finds the offer unacceptable and so counter-proposes. Since FSE is now engaged in multiple, concurrent negotiations, it selects a strategy that places a high degree of emphasis on the amount of resources it consumes during the course of the negotiation (variant of rule 2 in figure 12 of the companion paper). As in its negotiation with CHL, time is also an important factor, but the behaviour of the opponent is relatively less important since the agents have an ongoing relationship.

At time t_4, the vet customer agents have received and evaluated the latest offers from CHL. Again assuming they are unacceptable, they need to counter-propose another alternative. CIC is now running very short of time and so modifies its strategy so that remaining time is an even more dominant factor in its decision making (further application of rule 1 of figure 13 of the companion paper). FSE places greater emphasis on resources consumed (because of its concurrent negotiation threads) and since its concessions have been mirrored in CHL's proposal it places a slightly lower relative weighting on the behaviour of its opponent.

Also at time t_4, CHE receives the proposal coming back from FSE. This proposal will be fairly conciliatory in nature (because FSE does not want to expend too much resource on the negotiation and because time is starting to become pressing). To reflect this cooperative stance, CHE modifies its strategy to be slightly more reciprocal and slightly less driven by the time remaining. This has the overall effect of making its response conciliatory in nature.

At time t_5, CHL receives the counter proposals from the vet customer agents. It evaluates them and determines that it should accept the offer of CIC that has again conceded to most of its demands. CHL determines that it should not try and extract even more concessions since time is now running short. Having accepted CIC's offer, CHL rejects FSE's offer. When FSE receives the rejection, it chooses to accept CHE's latest offer because it wants to ensure its services are utilised and because the offer incorporates the majority of its demands.

In summary, CHL and CIC reach an agreement because they both concede throughout the negotiation (although CIC gives more ground). Consequently, they quickly reach the region where their acceptance levels intersect. On the other hand, FSE's rate of approach to the acceptance level intersection region is slower. The CHE-FSE negotiation is short because both agents concede quickly (due to the trust built up in their previous interactions) and so the agreement space is entered into early on in the negotiation. Finally, because CCL could not map to the information model required by CHL and because it was not CHE's most preferred service provider it failed to secure a deal.

4.3.2.2 Negotiating for the `Design_CPN` Service

This scenario covers the negotiation between the CHL and NDD agents for the `Design_CPN` service. Here we focus on the tactical and evaluatory aspects of the negotiation model and we assume that, once selected, strategies are not modified. The domain information about reservation values and issue importance is shown in figure 10. We consider two distinct negotiation situations: (i) both agents have a short time to reach an agreement (time for agreement is `NOW`) and (ii) both agents have no significant time pressure (time for agreement is `HAVE-TIME`). Since the scenario involves only two agents, reasoning about the logistics of the negotiation is trivial. As before, an agent's strategy is represented as a 3-tuple, indicating the relative importance of the time-, resource- and behaviour- dependent tactics.

	CHL	NDD
[$price_{min}$, $price_{max}$]: $weight_{price}$	[10,20]: 0.5	[18,30]: 0.4
[$volume_{min}$, $volume_{max}$]: $weight_{volume}$	[2,7]: 0.25	[1,3]: 0.4
[$penalty_{min}$, $penalty_{max}$]: $weight_{penalty}$	[2,5]: 0.25	[0,3]: 0.2

Figure 10: Reservation values and relative importance weightings for the `Design_CPN` service negotiation

Before the specifics of the scenarios can be discussed, a more detailed description of the operation of the tactics needs to be given. What follows builds upon the discussion in section 3.3.3.4 of the companion paper and the corresponding justifications for the particular formulae used are given in [13].

The time- and resource- dependent tactics operate in similar ways. Initially, they put forward a value that lies somewhere within their reservation range (see figure 14 of the companion paper for examples of the reasoning behind the setting of these values). They then modify this value in subsequent negotiation rounds depending on the amount of time or resource remaining. This continues until either the agent has an offer accepted or the time by when an agreement must be in place passes (in which case the negotiation has failed). At their deadline, the agents put forward their reservation value. More formally, in a negotiation between agents *a* and *b* over issue *i* (that has a minimum value of min_i and a maximum value of max_i for agent *a*), the value (x) put forward by *a* at a particular instant is determined by the negotiation parameter α ($\alpha \in \{0,1\}$) in the following way[5]:

$$x_{a \to b}[i] = \begin{cases} min_i + \alpha_i(max_i - min_i) & \text{if the reservation value is the minimum of the 2 values} \\ min_i + (1 - \alpha_i) \cdot (max_i - min_i) & \text{if the reservation value is the maximum of the 2 values} \end{cases} \quad (1)$$

From this, it is clear that the precise point in the range of acceptability is determined by the value of α. For time-dependent tactics, the value of α for agent *a* for a given issue *i* at a particular time *t* is computed in the following manner:

$$\alpha_i^a(t) = \kappa_i^a + \left(1 - \kappa_i^a\right)\left(\frac{min(t, tmax)}{tmax}\right)^{\frac{1}{\beta}} \quad (2)$$

where *tmax* is agent a's negotiation deadline, $\beta \in \Re^+$ is the decay function for the tactic (i.e. how quickly the rate of concession changes—the higher the value of β the more quickly the agent concedes), and $0 < \kappa i \leq 1$ is the value of the initial offer for issue *i*[6].

For resource-dependent tactics, α is based on the amount of resource consumed or remaining in a given negotiation episode. In this case, α is proportional to the number of agents actively involved in the negotiation at the current moment (NA) and inversely proportional to the number of messages (NM) that have been sent in the course of the current service provisioning episode. (This tactic has the effect of increasing the rate of concession as the number of agents being negotiated with decreases and the number of message interchanges increases):

$$\alpha_i^a(t) = \kappa_i^a + \left(1 - \kappa_i^a\right) \cdot e^{-\left(\frac{NA}{NM}\right)} \quad (3)$$

Behaviour-dependent tactics differ from time and resource related functions in that they base their behaviour on the actions of their negotiation opponent. In this scenario, agents exactly reproduce, in percentage terms, the behaviour of their opponent one time step ago. When insufficient exchanges have taken place to determine the way in which an opponent's behaviour is changing, the agents adopt a time-dependent tactic that is moderately conciliatory in nature (in order to maximise the likelihood of eliciting a cooperative response [3]).

As indicated in section 3.3.3.4 of the companion paper, the values put forward for a given SLA are a combination of the values suggested by the three tactic families for each negotiation issue, moderated by the relative weight set by the strategic reasoner.

Once the values for each of the negotiation issues have been set, the SLA is sent as a proposal to the negotiation opponent. The SLA is evaluated according to the following rules (section 3.3.3.2 of the companion paper). Assume agent *a* is the sender and agent *b* is the recipient. If the current time is greater than the deadline, then the offer is rejected and negotiation is terminated. Else if the utility of the received SLA is greater than or equal to that which *b* would send to *a* at this time, then *b* accepts the SLA; otherwise *b* offers *a* the SLA it generated for the comparison. Agents rate SLAs using a linear scoring function and then combining these scores using a weighted sum of the issues' importance. The rating *V* that agent *a* attaches to the value *x* of issue *i* in the SLA, is computed in the following way:

$$V_i^a(x) = \begin{cases} \dfrac{x - min_i}{max_i - min_i} & \text{If the value is acceptable and increasing} \\ \left(1 - \dfrac{x - min_i}{max_i - min_i}\right) & \text{If the value is acceptable and decreasing} \\ -100 & \text{If the value is unacceptable—ie } max_i < x < min_i \end{cases} \quad (4)$$

---

5. By means of illustration consider the negotiation over price. For CHL, min_{price} is 10 and max_{price} is 20 and the reservation value is the maximum of the two values. Whereas for NDD, min_{price} is 18 and max_{price} is 30 and the reservation value is the minimum of the two values.

6. Thus, for example, if CHL makes an initial offer of 15 for price, κ_{price} is 0.5.

With the relevant details in place, we can now return to the negotiation scenarios (figure 11). We first consider the case where both agents have a short time to reach an agreement. We assume there is a universal discrete clock, set to zero before the first proposal. Let the deadline to reach an agreement (t_{max}) be 8 time units for both agents. As CHL has only a short time to reach an agreement and because it is negotiating with an agent from the same organisation, it selects a strategy based solely on time remaining for each of the three issues (rule 3 of figure 12 of the companion paper). Thus its strategy can be represented as the following tactic weight vector [[1,0,0],[1,0,0],[1,0,0]] for the issues of price, volume and penalty respectively. The NDD agent goes through a similar reasoning process, but since it is the sole provider of a service it knows it can afford to negotiate a bit harder (choose a behaviour-dependent strategy) on those issues that it finds important. Thus, it adopts the following tactic weight vector [[0,0,1],[0,0,1],[1,0,0]] for the issues of price, volume and penalty.

Since CHL is under time pressure and since NDD belongs to the same organisation, it makes low initial offers on all issues (opposite of reasoning of rule 1 in figure 14 of the companion paper). In this case, “low” corresponds to fifty percent of the difference between the minimum and the maximum of the issues’ reservation values (i.e. κ is 0.5 for all issues). Also it sets a comparatively high concession rate for its time-dependent tactics ($\beta = 5$) since it needs to reach an agreement quickly. The utility of this offer to CHL is 0.5 (from equation 4: [0.5* 1 - (15-10)/(20-10)] + [0.25*(4.5-2)/(7-2)] + [0.25*(3.5-2)/(5-2)]).

On receiving this offer, NDD performs two concurrent operations. It evaluates the incoming SLA that has a utility of -100 (using equation 4). It also computes the offer it would have sent to CHL based on its current strategy. In this case, NDD’s initial stance is tougher than that of CHL and it selects twenty percent of the difference between the minimum and maximum of the reservation values for issues of price and volume (that it considers to be more important) and fifty percent for penalty (that is less important). Thus its initial offer is [27.6, 1.4, 2.49]. These values were computed in the following way. The time-dependent tactics suggest a relatively large concession over penalty ($\alpha_{penalty} = 0.83$; in equation 2: t_{max}=8, t=1, β=5, κ=0.5). From equation 1, the value offered for penalty is: 0.83.(3-0)=2.49. The behaviour-dependent tactics concede comparatively less on price and volume (a concession is made since the behaviour dependent tactics require at least two offers to be made before they can operate and in the meantime they concede a moderate amount—see above discussion). The utility of this initial proposal is 0.67 (from equation 4: [0.4*(27.6-18)/(30-18)] + [0.4*(1-(1.4-1)/(3-1)] + [0.2*(1-(2.49-0)/(3-0)]) which is greater than -100 and so NDD sends [27.6, 1.4, 2.49] to CHL as its counterproposal.

Upon receipt of this counterproposal, CHL computes its utility to be -74.95 (from equation 4). The value is negative since only the penalty issue is within CHL's range of reservation values. At the same time, CHL computes the counter-offer it would send at this time. The value for α_{price} is 0.88 (in equation 2: t_{max}=8, t=2, β=5, κ=0.5) which means the offer suggested by the time-dependent tactics is 18.78 (from equation 1). Since the time-dependent tactics compute α independently of the issue $\alpha_{price} = \alpha_{volume} = \alpha_{penalty}$. Combining the values suggested by the different tactics, the comparison offer is [18.78, 2.6, 2.36]. The proposal has a utility of 0.12 (from equation 4) which is greater than -74.95 and so it is sent as a counter-offer.

CHL’s new offer is now within NDD’s acceptable region on all issues (utility is no longer negative). However, the offer NDD would have sent at this time is [22.03, 2.42, 2.73] (calculations are as before). Note that in generating this offer the behaviour-dependent tactics reciprocate CHL's concession over price and volume (e.g. over price CHL has conceded by 25.2% and so NDD comes down by the same amount to 22.03). The utility of the proposal NDD would have sent is 0.268 which is greater than 0.147 and so NDD sends it.

The process of evaluation and counterproposal generation continues until one of the agents receives a bid that has a higher utility than the offer it was preparing to send. In this scenario, this happens at T=6 for NDD. This service provisioning episode reaches a successful conclusion because CHL uses time-dependent tactics that give ground rapidly because of the short time available in which to make an agreement. Moreover, CHL’s concessions are reciprocated by NDD since it uses behaviour-dependent tactics for the issues of price and volume. When taken together, these factors mean that the region of agreement is entered into quickly. The final point to note is that although the rate of concession of the two agents is almost identical, the end points of the utility functions for the two agents are different (0.065 vs. 0.18). This is because NDD took a tougher negotiation stance by making lower initial offers.

Time		NOW		HAVE-TIME	
		CHL	NDD	CHL	NDD
T=1	**offer** **utility**	[15.00, 4.5, 3.5] 0.5		[11,6.5,4.7] 0.9	
T=2	**utility** **new-offer** **utility**		-100.0 [27.6,1.4,2.49] 0.674		-100.0 [26.4,1.59,0.89] 0.70
T=3	**utility** **new-offer** **new-utility**	-74.95 [18.78, 2.60, 2.36] 0.12		-100.0 [13.72,5.13,3.88] 0.63	
T=4	**utility** **new-offer** **new-utility**		0.147 [22.03,2.42,2.73] 0.268		-100.0 [21.85,2.22,1.70] 0.37
T=5	**utility** **new-offer** **new-utility**	-49.92 [19.35, 2.32, 2.19] 0.065		-74.98 [17.3,3.34,2.32] 0.23	
T=6	**utility** **new-offer** **new-utility**		0.23 [21.39,2.71,2.86] 0.18 **accept**		-79.9 [20.19,2.64,2.41] 0.185
T=7	**utility** **new-offer** **new-utility**			-50 [18.67,2.75,2.21] 0.121	
T=8	**utility** **new-offer** **new-utility**				0.124 [20,2.7,2.41] 0.16
T=9	**utility** **new-offer** **new-utility**			0.07 [18.89,2.60,2.26] 0.11	
T=10	**utility** **new-offer** **new-utility**				0.1577 [19.98,2.71,2.44] 0.16
T=11	**utility** **new-offer** **new-utility**			0.07 [19.02,2.51,2.24] 0.09	
T= 12	**utility** **new-offer** **new-util-ity**				0.18 [19.7,2.75,2.55] 0.13 **accept**

Figure 11: Exemplar `Design_CPN` service negotiations where time is short (left) and time is plentiful (right)

In the second service provisioning episode, the two agents are not under such severe time pressure. Again, we assume that there is a universal discrete clock, set to zero before the first proposal. Let the deadline to reach an agreement (t_{max}) be 15 time units for both agents. Under these circumstances, the time-dependent tactic's rate of concession is slower than the previous scenario (since the interval to *tmax* is higher). Resource- and behaviour-dependent tactics behave as before since the former models concessions according to the number of agents and the length of the negotiation thread and the latter is independent of the time remaining.

All the domain information (reservation values and weights) is similar to the previous scenario. The first difference is in the strategies that the agents select. For CHL, although it has plenty of time for negotiation, it still needs to conserve its resources since it is likely to be involved in several concurrent negotiations. In addition to this, since it is negotiating with an agent that belongs to the same organisation, it should endeavour to reciprocate any concessions made by the other party. These two factors are of equal importance and are used by CHL to set [0,0.5,0.5], [0,0.5,0.5], [0,0.5,0.5] as its strategy for the issues of price, volume and penalty (see rule 4 of figure 12 of the companion paper). NDD, on the other hand, can afford to adopt a tougher negotiation stance (for the same reason as above) and so it places some weight on the time-dependent tactics (that give slow concession rates when there is plenty of time until the deadline). It must also take resources into consideration because it will be involved in multiple, concurrent negotiations. Finally, for similar reasons to CHL, NDD should reciprocate concessions at least to some degree. For these reasons, NDD adopts the following weight vectors for the three tactics ([0.1,0.6,0.3], [0.1,0.6,0.3], [0.1,0.6,0.3]) for issues price, volume and penalty (a variant of rule 4 in figure 12 of the companion paper). We again assume that strategies are static.

Negotiation is again initiated by CHL. Since there is a reasonable amount of time to negotiate, let the strategy of both agents be to set their initial offers to the ten percentile of the differences in an issue's reservation values for all issues (see rules 1 and 2 in figure 14 of the companion paper). The rationale for this is if the initial offer is further away from the reservation value, then it is more likely that the final offer will yield a larger utility (and the fact that the negotiation may take longer is not a major concern) [6].

Upon receipt of this offer, NDD evaluates it and finds it to be unacceptable since none of the values are within its reservation range. As before, it generates a counter-proposal by the same means as in the previous scenario. The weighted combination of tactics suggest 26.4 units for the price of the service. This value is computed in the following way. The time-dependent tactics compute the value of α_{price} to be 0.1 (from equation 2: t_{max}=15, t=1, β=0.4, κ=0.1). Therefore the suggested offer by this tactic family is 28.8 (from equation 1). Resource-dependent tactics compute the value of α_{price} to be 0.43 (from equation 3: κ=0.1, NA=1, NM= 1). From equation 1, the resource-dependent tactics suggest 25.6 for price. Finally, since CHL has not made at least two offers to NDD, the behaviour-dependent tactics concede a small amount (to 28.2) by playing a time-dependent tactic with a value for β that gives away a small amount utility. These three values are then linearly combined according to the weights that the strategy has selected (namely [0.1,0.6,0.3]). Hence the actual value NDD utters to CHL over price is 26.4 ([28.8*0.1]+[25.6*0.6]+[28.2*0.3]). Similar reasoning generates the other values for volume and penalty.

The process of evaluation and counter-proposing continues in this fashion until there is a cross over in the acceptance levels for all of the issues (time T=12 for the NDD agent). It can be seen that offers begin to become mutually acceptable after eight time steps (cf. sixth time step in the previous scenario). In addition to this, CHL's rate of concession for all issues becomes progressively less as time passes. This is due to two factors. Firstly, CHL is reaching it's reservation values for all issues because the resource-dependent tactics (with a weighting of 0.5 for all issues) suggest values close to reservation values as the length of the negotiation thread increases. Secondly, the other, and equally weighted component of the CHL's offers, are the behaviour-dependent tactics. Since NDD is conceding less (compare offered utilities in T=9 column and T=11 column) these tactics imitate this resistance to concession by the opponent on all issues and suggest offers close to, or at, the values of CHL's previous offer. Therefore the dynamics of CHL's offers are the resultant forces of these two tactics—resource-tactics suggest concession and behaviour-tactics suggest firmness in the light of NDD's reluctance to concede.

4.4 Service Management

During negotiation, it is important that an agent possesses the latest information about its planned resource usage so as not to over commit the business unit it represents. ADEPT agents maintain a resource model and a schedule of resource commitments that enables them to determine whether they can support an SLA. In the vet customer negotiation, for instance, the CIC agent needs to consider whether it has sufficient instances of the `vet_customer`$_3$ task (see figure 2) to provide the `Vet_Customer` service under negotiation (i.e. if it has enough resources to complete the agreement). By evaluating the availability of its resources over the period of time under negotiation, the CIC agent can deduce whether it will be able to fulfil the agreement. If there are insufficient resources, the CIC agent may reject the proposal or counter-propose an alternative time period for completing the service. The resource model itself is dynamic. Resources can register and de-register their availability with their controlling agent. For example, if a person in the `Vet_Customer` team is off sick, their resource information is de-registered (temporarily lowering CIC's capacity) and then re-registered when they return to work.

ADEPT agents differentiate between those SLA's that cover one-off services and those that cover on-demand services. One-off services can be scheduled accurately by reserving the appropriate resource for the agreed period stated in the SLA. For example, after an `Analyse_CPN` negotiation is completed, NDD is able to schedule one invocation of the `analyse_requirements` task over the time period defined within the negotiation. On-demand services, on the other hand, require the block booking of resources to cover multiple invocations over a certain period. In this case, an estimation of the volume of expected executions is required (this comes from the volume slot in the SLA—see figure 7). The ADEPT agent then sets aside an appropriate number of resources (task instances) for a particular period of time. The quantity and duration of these reservations are influenced by the agent's self knowledge about the maximum number of concurrent invocations and the average time a service takes to run. When an on-demand service is actually invoked, the agent simply uses part of the associated block booking it has already set up. Consider, for instance, the case of CIC forming an agreement with CHL for the `Vet_Customer` service (section 4.3.2.1). If CHL negotiates well, it may end up with an SLA that has a volume of 40 invocations between 8:30 and 18:00 (CIC's greatest capacity—figure 7). In this case, CIC needs to reserve sufficient instances of the $\texttt{vet\_customer}_3$ task to fulfil this responsibility.

The resource management function that is presently performed by the ADEPT agent could be outsourced to an existing legacy resource management system (as discussed in section 3.1 for the legal department). In this situation, the existing system would schedule and allocate work and provide resource utilisation information to the agent during negotiation. This approach might be necessary where resources need to be optimised across a business unit, or where there is a heavy reliance on a legacy system, or even where complex estimation calculations are required that would be better sited outside the agent.

The other main service management functionality relates to handling exceptions. Here there are four classes of exception that can occur during the service management phase:

1. *Communication failure*: Information sent between agents becomes corrupted. For example, the message informing CHL of the result of CIC's vetting may become corrupted. This would be detected by CHL's communication module that would automatically inform CIC of the problem. On receiving this message, CIC's communication module would simply resend the message.
2. *Functional failure*: When a single invocation of a task fails. For instance, an error is made by a $\texttt{vet\_customer}_3$ resource. Although the resource is still registered, the vetting of the customer needs to be re-evaluated. This is undertaken by CIC selecting another free, and functionally equivalent resource (task), and invoking it with the original information.
3. *Resource failure*: A problem occurs with a domain problem solving resource. For example, one of NDD's network designers becomes unavailable and so NDD's ability to complete the `Design_CPN` service is reduced. In this case, the agent tries to reschedule the tasks that were assigned to that resource (designer) without violating any of the existing SLAs.
4. *Service exception*: An agreed SLA is in danger of being violated. Continuing the above example, because of the absence of the designer, the NDD agent may be unable to fulfil all its obligations on time (it simply lacks the manpower). In this case, NDD will attempt to proactively renegotiate the SLA (perhaps accepting a penalty) before the violation actually occurs. Only if this renegotiation fails will a service exception be issued.

It can be seen that the agent attempts to handle these exceptions at various levels. The least serious ones are dealt with locally by repeating or re-assigning tasks, the medium ones by re-scheduling tasks, and the most serious ones by social interactions.

5 INTERFACE DESIGN ISSUES

Agent-based systems with their emphasis on autonomy, pro-activeness and reactivity offer a level of flexibility and parallelism not usually seen in complex system developments. In the provide customer quote application, for instance, many of the sequential dependencies that abound in conventional systems were removed. Thus users could instruct sub-systems to carry out tasks that are not possible at the instant of invocation, but, later, when conditions change, these subsystems can automatically react and commence the previously requested task. This means, for example, that at the commencement of the provide quote business process, the customer handling agent can request customer vetting whilst simultaneously dealing with the task of gathering the customer's initial requirements. The vetting process might involve the vetting agent accessing multiple, external databases and interpreting their results. This process will invariably involve a degree of latency. However in the agent metaphor, the activity is left to the agent responsible. When the task has been completed, it reports back the results. This pattern can be observed throughout the provide quote application.

Whilst this degree of flexibility and parallelism enables powerful systems to be constructed, it raises a number of new issues and challenges for the design of the human computer interface [10]. In this section, we identify and discuss these issues and then present the solutions devised for this application. Note, however, that although these issues arose out of this specific application, we believe they are relevant for the design of the human computer interface in most agent-based systems.

When designing and building the provide customer quote interface, four basic problems were identified [9]:

- *Impenetrability*: users cannot determine why a particular set of actions has happened (or has not happened). Many of the system's actions are hidden inside the inter-agent negotiation process.

- *Data Handling Difficulties*: users are not familiar with the sources of the data, its reliability and its accuracy. Problems are also caused by the sheer volume of the data involved.

- *Process Complexity*: the autonomous operation of agents, the complexity and speed of the negotiation process, and the distribution of problem solving across the system all combine to form a complex environment that even the system designers have difficulty in following.

- *Accountability*: it is difficult for a user (or a designer) to know that the system has performed as expected and yet there is a requirement that the system can be audited and monitored.

The causes of these basic problems vary between applications. However in this case they are as follows. Impenetrability is caused by a lack of appropriate feedback at an appropriate level (and this level will depend upon the type of user and the nature of the task being carried out). However, too much feedback can be as perplexing as too little feedback. Data handling problems occur because the sources of the data may be unknown or unfamiliar at the outset of the service. Agents may search the organisation for appropriate sources of data and supply too much or too little, or data of the wrong type. The complexity problem is related to the combinatorial explosion that arises out of the interactions between a set of autonomous problem solving entities. There are a huge number of possible problem solution paths that can be followed, so some form of abstraction is clearly necessary to tackle the complexity. The accountability issue is an important one. How can users or designers obtain an account of what the system has done and why? Who should be blamed if the system delivers an inappropriate or inefficient solution? How can such problems be corrected? The accountability problem resonates with similar problems that came to light early on in the development of expert systems—who is responsible for a faulty diagnosis?

Each of the aforementioned problems manifest themselves in different ways and to different degrees to the different types of user in the system. In this application, however, we were particularly concerned about impenetrability, complexity and accountability, though problems associated with data volumes sometimes occurred when the system was fully loaded. Since the type of user is important, we divide the subsequent discussion into the three classes of user found in this application: (i) *end application users*: individuals who perform problem solving tasks in the business process (e.g. input clerks, design engineers, etc.) (section 5.1); (ii) *system designers*: programmers and analysts who write and maintain the ADEPT system (section 5.2), and (iii) *business process managers*: who are responsible for monitoring and managing the end to end business process (section 5.3). For each type of user, we discuss the most pressing issues with respect to interface design. Section 5.4 then describes the philosophy of the ADEPT approach to tackling these issues for agent-based systems in general. Finally, section 5.5 describes how these issues were addressed within the provide customer quote application.

5.1 Usability Problems for the End Users

The users of the provide customer quote application form a spectrum ranging from input clerks and secretaries to engineers, lawyers and senior managers. They certainly have one thing in common—a lack of, and a disinterest in, computing knowledge for its own sake. Some will almost certainly exhibit computer phobia. At one time it was common to talk of computer application users as being "naive". Other classification systems talked of them being "beginners" or "experts" [7, 12]. However, this can be misleading and trivial. All computer users are usually experts in the area of their professional competence, and they expect others, with different expertise, to converse in a manner that will convey the required information without resorting to unnecessary technical knowledge or jargon. A BT lawyer frequently has to converse with a BT engineer, but neither expect to require special support to achieve this. Of course they may have to explain certain situations and procedures to the other party and check for agreement, but the interaction is usually achieved through mutual support. Computer applications (and agents) should operate in a similar way so that the professional user can get on with their task without having to carry out additional, and usually irrelevant tasks, because of a poor user interface. Here the key point is to provide feedback at the right level and at the right time.

The second major problem experienced by this class of user relates to tracking the status of problem solving tasks in a decentralised organisation. Users want answers to questions such as: which agent is dealing with my problem?, who had it last?, why is this issue currently on hold in the Legal Department?, what are network design team waiting for?, and why has it not progressed? These problems are not new and are often experienced in manual systems as well. Indeed in a manual system, there is eventually someone who can track down what has happened and take corrective action, and we needed the equivalent in our automated application.

The final problem for this class of user relates to determining the degree of autonomy exhibited by the agents. Professional users, like network designers and lawyers, often wish to influence certain aspects of the system's operation: do not do that now, give more priority to this, prod the customer handling department to speed things up, etc. These wishes need to be reflected in the interface presented to users, if they are to be happy about their roles in the business process relative to those of the agents. It must be made clear that the agents are the servants of the human users, not the other way round. Achieving this requires a clear definition of the limits of responsibility and delegation between agents and users. When human beings act as agents to other human beings, they only do so within well defined limits. There is an "accepted", if sometimes implicit, contract between the person providing the service and the agent servicing the request. This contract defines the limits of delegation. This "service contract" has certain similarities with the service level agreements outlined earlier, but it is a contract between a human user and a machine-based agent. Such service contracts are required at the interface between humans and the system particularly where the service is not well defined. For a more extensive discussion of this issue see [1].

A general point underlying all of these usability issues relates to the domain language used by the agents to interact with their users. In all cases, users want to know the answers to their questions in the domain language that they are used to. For example, a lawyer will not wish to know which software agent is in control, rather, what is the state of the negotiation? or what is holding up the resolution of the problem? The implications for systems design are far-reaching. It means that different levels of abstraction will need to be held simultaneously, and updated, connecting the operations of the low level agents with the higher level views.

5.2 Usability Problems for the Designers

Distributed systems that have a high degree of autonomy and pro-activity are amongst the most difficult to test and debug [8]. The problems are, of course, very similar to those described above for the system's end users, only they are at a different abstraction level, being concerned with technical detail rather than business detail. To this end, the type of services that designers require during the system development phase were:

- a what-if facility to allow exploration of alternatives in agent activity (e.g. what happens if we change the SLA between the CHL and CIC agents for the `Vet_Customer` service?);
- a what-happened facility at a detail level (e.g. why was the `BT_Customer_Credit_Rating` so low for this customer? Which vetting process was used?);
- a facility to "sit in" on a particular agent and direct its negotiation strategy (e.g. actually take the place of the agent by constructing message replies during negotiation with other agents);
- identification of errors from agent interactions (e.g. by "sitting in" on an agent negotiation process and watching the progress of a suspect negotiation)
- analyses of possible inconsistencies arising from agent actions, both from the design, and from actual execution (this requires some form of formal analysis technique, perhaps by labelling paths and then using algebraic techniques on path combinations)

5.3 Usability Problems for Business Process Managers

Finally, we observed that there are also difficult problems for those who managed and monitored the overall system operation. How do managers optimise the operation of such systems? That is, how do they identify and remove bottlenecks, identify redundant procedures and eliminate inefficient solutions? How can the system support them in these actions? How is accountability in the legal or financial sense dealt with? This "management" accountability problem requires specialised knowledge as previously described for the two user groups above, but this knowledge will be more statistical and historical in nature, since the prime objective is to compare previous system interactions and timings with current activity.

5.4 Tackling the Usability Problems: Adding Self Knowledge to Agent Systems

We tackled the aforementioned usability problems by adding additional capabilities to the agents so that they had self-knowledge about their operations. With such knowledge, the system is able to reason about its past, current and future actions, becoming more accountable to all classes of user. Examples of the types of self-knowledge we added relate to: knowledge of the system's current overall state, how it was reached, why it arrived there, and where it might go next. This knowledge can then be interrogated by the user (of whatever level) in order to satisfy queries and concerns. Since the different classes of users have different expertise and different knowledge requirements, we had to store the knowledge at different levels and provide a different emphasis for the different users. Thus we store additional knowledge about the agent activity at different levels. An example of self-knowledge at a high level of abstraction is the business processing flow diagram (similar to that shown in Figure 1). Without a correspondence between low level agent activity and this high level processing diagram, the user cannot be advised in business terms.

One problem that taxed us was how to ensure that the user could understand the agents' self knowledge. The key to this is presenting it in the most appropriate form. This is a separate problem from deciding the most appropriate level of abstraction, though, of course, there is a connection. One clue as to how to approach this problem came from work done in trying to validate rule-based expert systems. In 1988, the European Space Agency was concerned about developing on-board, rule-based expert systems in satellites. One approach which proved promising was a combination of qualitative modelling of the operation of the satellite, a rule induction approach used to derive a set of expert system rules, and a visual model of the system which could be observed by the engineers [2, 11]. What turned out to be important was the visualisation of the qualitative model, and the ability to exercise the model under different conditions. By watching visual presentations of the execution of the model (actually driven from the basic model components), they were able to confirm the validity of the models and understand the consequences of errors.

There is a similarity here because an agent system can be interpreted from both qualitative and quantitative points of view, the former is more appropriate for end users and the latter more appropriate for designers. Thus, by setting up different visualisations of the operation of the model (both qualitative and quantitative), we can present the self knowledge of the system in representations that can be understood at multiple different levels. Out of this idea arose the key concept of the *agent visualiser* that was used to solve the problem of communicating the self knowledge at the right level of abstraction.

5.5 The ADEPT Visualiser

The visualiser is essentially another agent in the system that negotiates with the application's main problem solving agents before operations commence, sets up information collection agreements (that can be thought of as SLAs), and then communicates with users during the operation of the system, visually presenting the supplied information in a variety of forms on request. It also records all the interactions so that they may be played back separately at a later time. In the current ADEPT implementation, a user can see two basic views:

- the agent view (service provisioning and service management activities),
- the business process view (the state of the system in high-level business terms).

Both views are driven from the same basic execution information. They show the current state in diagrammatic form and messages from active agents alter the state of the diagram. For end users (lawyers, network designers, etc.), a figurative view of the whole business process is presented and progress is marked by colour changes (figure 12). For agent designers, actual agents are shown with the content of the messages passing between them (figure 13).

In both figures there are a number of common items. At the top left of the screen are three control menus for selecting various strands of information (a strand being a negotiation for a service, a service execution, or a business process). For a given strand, the viewer can select a thread (a particular business process diagram), the sub-thread (e.g. `Capture_Customer_Details`) and the message content. Below this is an expansion panel that shows the content of each message. Then there is an expansion of a particular message and below this the current SLA of interest. Control buttons at the base of the screen allow the viewer to navigate, follow progress or look at previously stored sequences. The right of the screen differs for the two views. In the agent view, there is an agent window and a negotiation window. When inter-agent negotiation takes place, arrows appear between the relevant agents and the label on the arrow contains the actual message (as can be seen in the figure). Double clicking on an agent allows the viewer to send instructions to agents as to what messages to relay to the visualiser. Even agents that have not been requested to send any messages, send skeletal messages to the visualiser so that it knows all the executions that have started. In the business process view, the top half of the screen shows a diagrammatic view of the whole business process. This is updated from agent activity and different colours indicate the different possible states of the processes (flashing green indicates active, red completed, etc.). Clicking on a process reveals a further breakdown into the flow diagram for this service. The lower half of the business view shows the agents that are executing in relation to a chosen process.

Although it has not been possible to carry out detailed experiments to precisely quantify the Visualiser's benefits (such experimentation is very difficult to successfully carry out), we have observed designers endeavouring to debug agent problems. This observation (and subsequent comments from designers) lead us to conclude that visualisation is an essential tool for high productivity. Moreover, it is our expectation that as the complexity of the system grows, so the importance of visualisation will increase (probably super linearly!).

In conclusion, the Visualiser has helped users in the following ways:

- The Business Level Representation has enabled *End Users* to locate what is going on and to understand how the current situation was reached;

- The detailed agent negotiation level has enabled *Designers* to watch the negotiation process and has assisted with debugging. The effect of SLAs has also been observed;

- Although the current Visualisation representation for *Managers* is not fully implemented, areas of slow activity and bottlenecks can be observed and monitored.

6 CONCLUSIONS

This paper has shown how ADEPT can be used as a solution technology for a real-world business process management application. We have illustrated how the conceptual framework described in the technology companion paper can be used in practice. In fact, a modified version of this application technology is planned for field trials within BT in the coming year. This experience has led to a number of important insights on agent-based business process management in particular and practical applications of multi-agent systems in general.

In terms of agent-based business process management, the ADEPT approach enabled us to develop a robust and flexible system that could not have been built using extant workflow technology[7]. Features not supported by traditional systems include: managing business process enactment in an environment in which the resources fluctuate at run-time, handling exceptions in a context-dependent manner, provisioning problem solving resources according to prevailing circumstances, and allowing loose coupling between inter-organisational business activities. Moreover, the agent-based approach enables different parts of the organisation (and indeed different organisations) to retain their autonomy of information and control. This is an essential feature for any technology that is to become widely accepted and used in this domain. To develop a system with these characteristics a number of important components needed to be put in place. These include a flexible and high-level means of specifying services that gives the agents sufficient freedom to take alternative paths at run-time, a fast and efficient means of sharing information between agents with different data models, negotiation strategies and tactics that can be tailored to the provisioning task at hand, and flexible mechanisms for scheduling and re-scheduling problem solving resources. The main drawback of the ADEPT solution is that it can be more difficult for the business process manager to track and monitor the end to end process (since it is now a highly decentralised and concurrent system). To alleviate this

7. Ideally, the benefits of the ADEPT approach would be shown quantitatively by comparing our system with a similar one developed using conventional techniques. However, with organisation-wide systems, such as ADEPT, such experimentation is extremely difficult because of the sheer scale of the system (it is estimated that a realistic comparator would take several person years to develop). Given this fact, the argumentation for ADEPT is more qualitative in nature.

problem, considerable emphasis was placed on developing a powerful suite of visualisation tools that enabled users to view the system from many different perspectives and at many different levels of abstraction.

The work also highlighted a number of broad issues that are relevant for the development of industrial strength multi-agent systems in general. The first relates to the importance of a multi-view suite of visualisation tools. If end users and developers are to understand the operation of a complex multi-agent system then it is essential that a range of powerful visualisation tools are available. Here we highlighted a number of views that we feel are essential to this endeavour. Secondly, efficient development of industrial strength systems requires a number of high-level facilities and structures to be available for building both the micro-level aspects of the system (i.e. the individual agents) and the macro-level aspects (i.e. the interaction structures and reasoning models). Such facilities are needed because we believe most applications will be composed of heterogeneous agents that are required to exhibit a range of problem solving characteristics. Given this, the most effective way of building such varied agents is to have high-level configurable components. Finally, heterogeneous information models must be treated as a given (even when developing a multi-agent system for a single organisation). This means the information sharing infrastructure must be an integral part of the system, not something that is bolted on later.

For the future, there are a number of lines of research and development that are being pursued. On the application side, the ADEPT concepts and software are being applied to a number of new business process management applications both within BT and within other large organisations. On the technology side, a number of open research issues still remain. Firstly, more elaborate forms of visualisation are being experimented with in order to complement the techniques developed thus far. These techniques include auralisation and virtual reality. Secondly, a more systematic evaluation of the behaviour of the negotiation capabilities is being undertaken. In the provide customer quote application, a significant amount of fine tuning of the negotiation strategies and tactics was required to obtain the desired performance characteristics. We wish to place this effort on a sounder footing and to formally show properties about the convergence (or not) of the various types of negotiation. Finally, this work has highlighted the clear need for a systematic methodology for analysing and designing multi-agent systems. The present system was designed based on intuition, past experience and, to a certain extent, trial and error. This is not sustainable for future developments and so work on multi-agent system methodologies is urgently needed.

REFERENCES

[1] J. L. Alty (1997), “Interface Agents and Dynamic Processes”, *Proc. 16th European Annual Conf. on Human Decision Making and Manual Control, (ed. Johannsen, G.,),* Kassel, Germany, 197-200.

[2] J. L. Alty, and D. A. Pearce (1992) “Artificial Intelligence and Fault Diagnosis: An Approach to the Validation Problem” *Proc. Int. Conf on the Applications of Artificial Intelligence to Industry*, Leningrad, Soviet Union, 121-129.

[3] R. Axelrod (1984) *The Evolution of Cooperation*. Basic Books.

[4] M. Bradley (1994) “Starting TQM from ISO 9000” *TQM Magazine*, 6 (1) 50-54.

[5] J. Bulled (1996) “ISO 9000 - A Route to Total Quality” Proc. of 1st Int. Conf. on ISO 9000 and TQM.

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

[7] J. Fisher (1991) “Defining the Novice User”, *J. Behaviour and Information Technology*, **10** (5) 437 - 441.

[8] L. Gasser, C. Braganza, and N. Herman (1987) “MACE: A Flexible Testbed for Distributed AI Research” in *Distributed AI* (ed. M. N. Huhns) Morgan Kaufmann.

[9] M. Lunnon and J. L. Alty (1997) “Visualisation of Intelligent Agents”, *Proc. 3rd ILOG Int. Users Mtg*, Paris, France, 34 - 42.

[10] D. A. Norman (1994) “How people might interact with agents” *Comms. of the ACM* 37 (7) 68-71.

[11] D. Pearce (1988) “The Induction of Fault Diagnosis Systems for Qualitative Models”, *Proc. 7th National*

Conf. Of Artificial Intelligence (AAAI-88), St. Pauls, Minnesota, 353 - 357.

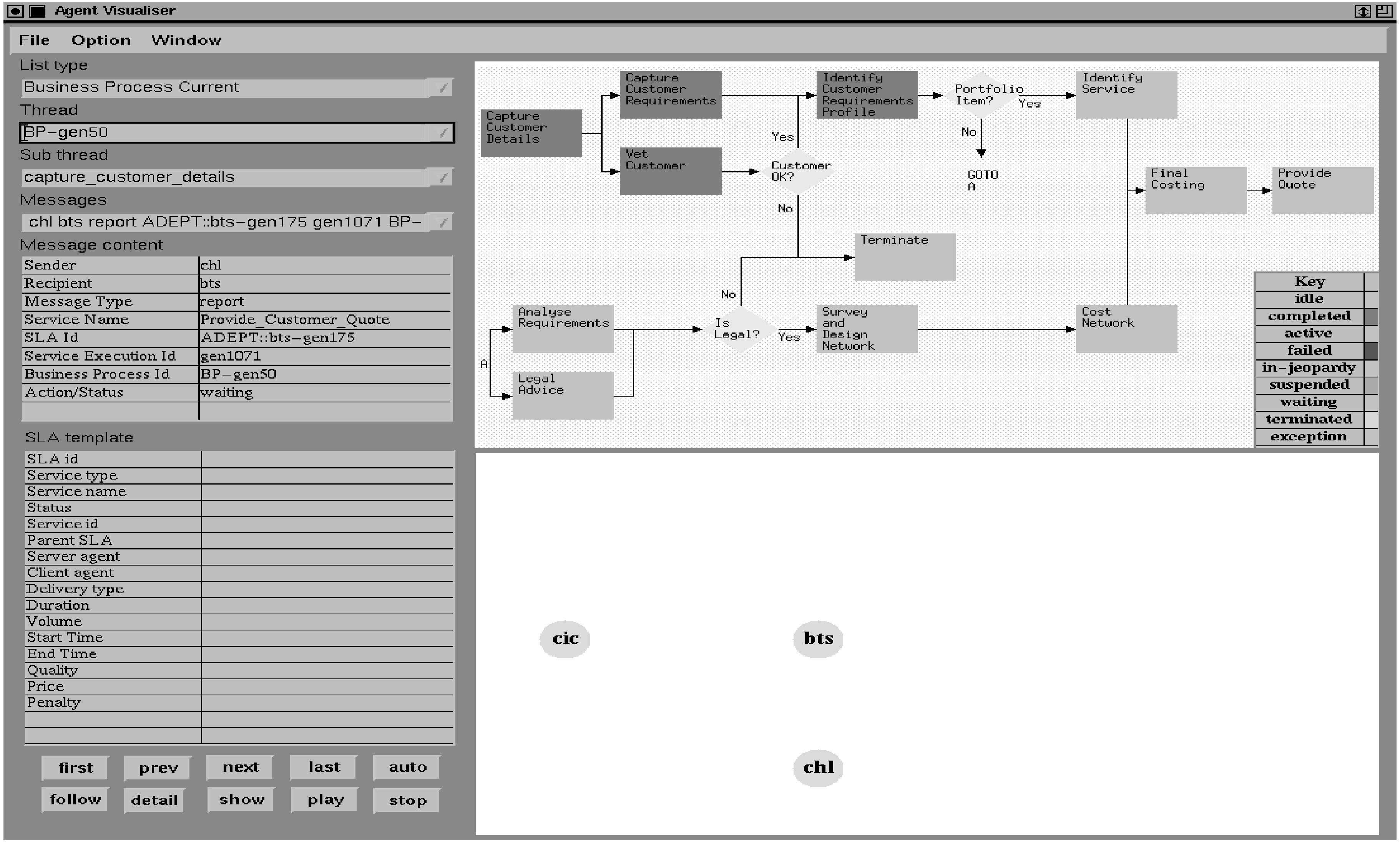
[12] C. Ryan (1992), "Usefulness of On-line Help to Novice Users in Supportive Learning Environments", *Proc. CHISIG Ann. Conf., on HCI Education*, 166 - 173.

[13] C. Sierra, P. Faratin and N. R. Jennings (1997) 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.

[14] I. Sommerville (1992) "*Software Engineering*" Addison Wesley.

[15] M. Wooldridge, N. R. Jennings, and D. Kinny (1999) "A Methodology for Agent-Oriented Analysis and Design" *Proc. 3rd Int Conference on Autonomous Agents* (Agents-99) Seattle, WA.

Figure 12: Visualiser's Business Process View



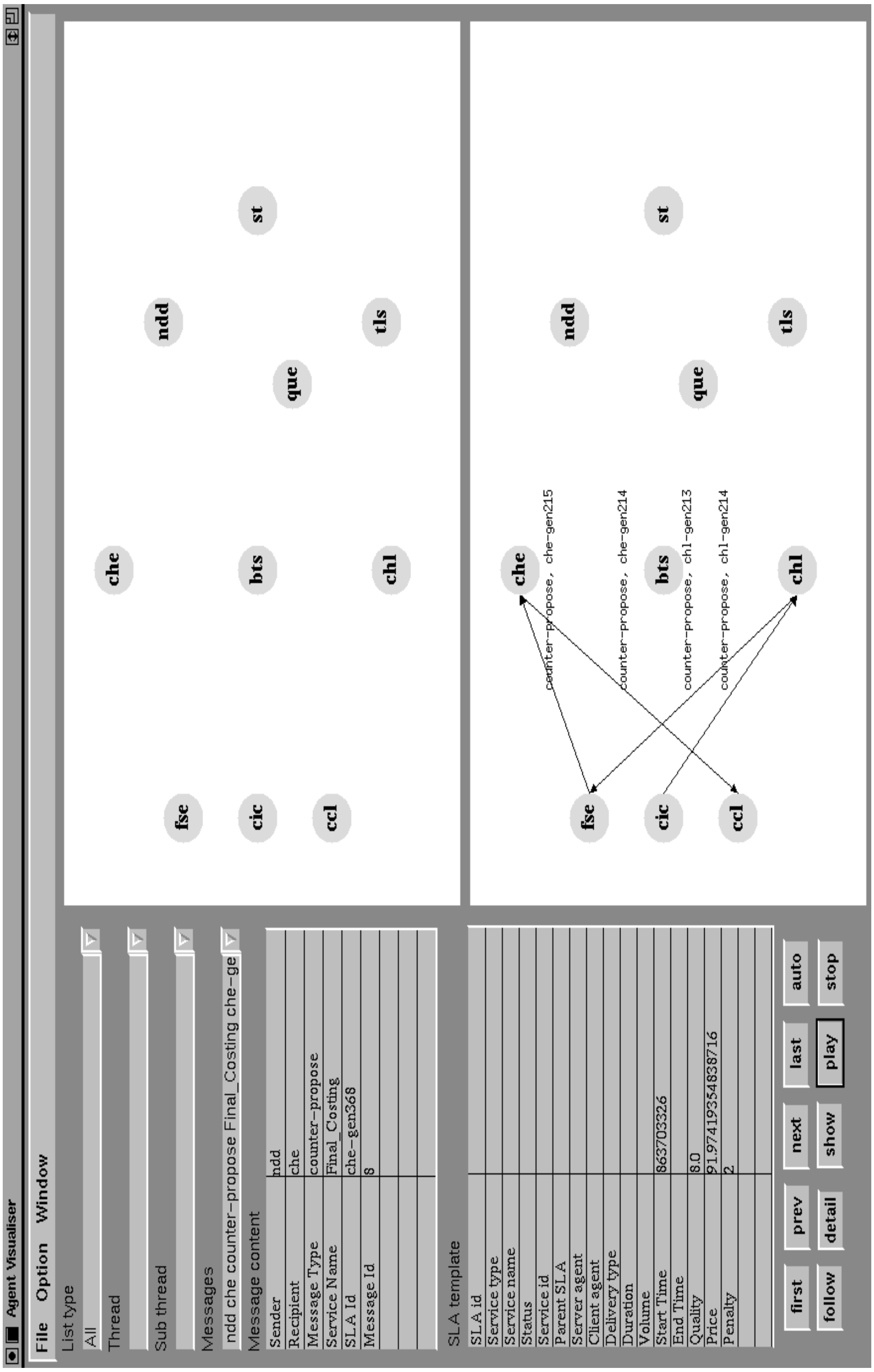


Figure 13: Visualiser's Agent Designer View