Intelligent Agent Framework for Order Entry and Management

Thuc Duong Nguyen
British Telecommunications Plc
Ipswich, UK.
duong.nguyen@bt.com

Simon Thompson
British Telecommunications Plc
Ipswich, UK.
simon.2.thompson@bt.com

ABSTRACT

This paper describes an agent system we have built to handle order entry and management issue in business computing and specialized in the telecommunication domain. Our system is implemented using only industry approved standards and it can both manage complex workflows as well as handle user context dependant service selection. Furthermore, it has the abstract goal representation mechanism, which in turn can detect and reflect to changes in its working environment, leading to more accurate capturing of user's requirements. We also detail a number of prototype applications that has been built using our technology and highlight how our approach addresses the business issues around each of these applications.

1. INTRODUCTION

Agent technology and web services are two key components for business computing problems [3, 8]. Agents are efficient for their ability to decide which actions to take independent of the environment that they are situated in [6]. Web services are loosely-coupled distributed systems that can be provided over the net, which allows businesses to have easy and agile interaction with their partners or customers. Combined together, they provide a concrete and robust business oriented problem solving solution [10].

Order entry and management is a hard business computing problem due to its complexity and the strict process that need to be followed. Consider an example of the problem of handling customer orders for a BT Broadband-Delivered Private Data Service product (PDS). This is a typical product that BT offers to a SME user. Here, the customer has two offices that need to be connected using a dedicated SDSL connection and BT will be the provider for hardware implementation. In order to handle an order of this type, there are a number of steps need to be taken, including:

- Based on the given customer office locations, use the online address checker to determine the correct exchanges. Connect to a central nodes database to determine the correct loop back nodes for these exchanges.
- If these locations are SDSL capable, there will be a number of DSLAMs at each exchange. The details of the DSLAMs are kept on a distributed database and are updated daily. From these DSLAMs, select ones with enough ports and bandwidth to have the connection established. If no such DSLAMs are available, either order a new one or cancel customer order. There are a few factors need to be considered in the selection including the current booking ratio, potential future expansion as well as the lead time of up to 30 days for a new DSLAM to be installed.
- If there exist suitable DSLAMs, the next step is to check if the connection between these exchanges and their loop back nodes can be obtained. Since it is a dedicated line, the loop back node must also have the spare capacity to handle the request. If this is not feasible, the customer order must be canceled.

PDS is just one single product in a broad portfolio that BT is currently offering. For each product, the procedures involved in handling customer requests are complex and error-prone. Currently most of these steps are manually handled by specialized customer agents, thus, leading to a significant increase in both time and resource required as well as the number of customer order being incorrectly processed.

Consider another related business scenario: handling a request from a SME user to relocate some of its offices to new premises. This type of order typically involves a large number of tasks including: disconnect existing services such as: electricity, intranet, phone ... and reconnect them at the new premises; physically move assets from one location to another location, etc. Besides, the orchestration of these tasks is also complex and requires careful monitoring. Multiply this with the number of offices, the services and people involved and the magnitude of complexity of the problem becomes clear.

Against this background, we have developed an agent platform called Kreno to tackle such complex order entry and management problems [10, 9]. The activities required to fulfill an order are considered as services and for each problem presented, our Kreno agent is able to select the a subset of these services and compose them in an order to provide a solution for this particular problem. Specifically, a Kreno agent consists of three main components [10]: (1) a planner to find the necessary services and put them in the right order to be executed, (2) a scheduler to convert the generated plan to a concrete workflow with the detailed running times for each of the services together with its resource requirement and (3) an execution engine to execute and monitor the workflow and to handle any potential exceptions. A developer interface is also included for the developers to design the interfaces to communicate between the web services.

The remainder of the paper is organized as follows: section 2 discusses the background of our work, section 3 details our Kreno

AAMAS'07 May 14–18 2007, Honolulu, Hawai'i, USA.

978-81-904262-7-5 (RPS)

framework, including the interfaces and the working mechanisms of the components. Section 4 presents the prototype applications that we have implemented using our framework and, finally, section 5 concludes with our evaluation of the usefulness of this approach.

2. BACKGROUND

A general business problem such as the one mentioned in section 1 is typically complex [9].

Order entry and management is a complex issue. At each step, the number of selectable choices can be large and each choice could impact on the subsequent selections and each individual choice could provide an impact on the future status of the system[1]. As most of the process is currently being handled manually, a rational selection is not always assured because of human error. Furthermore, the number of errors and time required will also be increased. Nonetheless, there are business rules to handle these various situations and if can be applied automatically, errors and the time consumed will be greatly reduced. However, managing and utilizing such rules present different problems.

The combination of web services and agent technology provide a potential solution to this problem [8, 4]. Intelligent agents, which are computer programs working toward goals in a dynamic environment without the need to have continuous supervision or control and exhibit a capacity in seeking to transform goals into necessary actions [6]. Web services, on the other hand, are self-contained modular business applications that have standard interfaces. In contrast to traditional client/server approach, web services are loosely coupled and not tied to any set of programming languages. Instead, they typically are message-driven and language independent and can be easily interacted with [3].

However, realizing a service oriented computing model is complicated [8, 3]. For example, there could be potentially hundreds of different services from a variety of vendors and even though they can inter-communicate, selecting the right subset of such services to handle a particular task is not straightforward. Plus, it cannot be guaranteed that a service will be available though out the life time of a task (from the planning to the execution).

To tackle this problem, we propose a solution in which web services are represented using STRIPS representation [2] and Graphplan [1] is used as the planning engine to find the rational subset of services (a plan) to handle customer requests. The advantages of this approach is that various services from different vendor can be represented generically and the planner can select the appropriate services without understanding underlying architecture of each and individual service. Later on, this plan will be converted to a concrete workflow that can be executed to provide a complete fulfillment to a particular customer order. Here, the issue of availability of the services is dealt with using a resource based scheduler. The details are described in section 3.

3. OUR KRENO FRAMEWORK

Our Kreno agent uses knowledge base (could be dynamically updated at runtime) to describe its environment, some goals might be achieved, and the interfaces to various web services are defined. These elements are the core of our Kreno platform and they are represented by sets of propositions using STRIPS representation [2]. A proposition consists of a main statement and a number of correspondent atoms in which the statement specifies the relationship among the atoms. The atom can be either a variable or a fully instantiated object. The following examples illustrate the definition:

- *isa(exchange, IPSW1E)*: is a standard proposition to indicate that atom *IPSW1E* and atom *exchange* has a relationship of *isa*. In other words, *IPSW1E* is an *exchange*.
- *isa(exchange, ?someExchange)*: is an extended proposition to indicate that atom *exchange* and variable atom *?someExchange* has a relationship of *isa*. Proposition *isa(exchange, IPSW1E)* is an instance of this extended proposition.

In Kreno, web services are the interfaces to tasks or services that carry out particular functions. Each web service is represented using three sub interfaces:

1. The *precondition interface*: states what propositions need to be satisfied in order for the service to be invoked. For example, the precondition interface for a service that is used to buy bandwidth for an user may include *have(user, money)* to indicate that the customer needs to have money for this transaction to go forward.
2. The *add effect interface*: specifying what propositions will hold after the service has been executed. Typically, it details new propositions generated. With regard to the previous example, the add effect interface may include *have(user, bandwidth*) to indicate that bandwidth has been reserved for that user after the transaction.
3. The *delete effect interface*: specifying what propositions will no longer hold after the service has been invoked. Typically, it details which propositions will be deleted. For example, the deletion interface of the previous example may include *have(user, money*) to indicate that after the bandwidth has been bought the customer will have no money.

Next, the core knowledge base for the agent to work on are also represented by sets of propositions with the restriction that no extended proposition is allowed in the knowledge base. Unlike other planning system (e.g. JASON [5], SPARKS [7]), the goal specifications in Kreno are abstract, meaning that the developers does not need to explicitly state the goals for the agent. Instead, an abstracted version can be specified and this will be compiled during the agent runtime based on the knowledge base to give more concrete goal instances.

The concept of a Kreno agent with its interface is displayed in figure 1. It is design based on industrial approved standards including: WSDL [11] / SOAP [11] for web services, WS-BPEL [12] for workflow specification, OWL / XML [11] for knowledge representation, JSP / J2EE for system and frontend implementation.

Breaking down, a Kreno agent consists of a number of elements, which include:

- *The user interface*: to allow developers to define communication interfaces with web services, end users to specify targets, modify their profiles (knowledge base) and interact with the system.
- *The internal reasoning and planing elements*: Kreno has built in planner and scheduler module to effectively compose a workflow to satisfy the user's targets.

[1]This could happen in both positive and negative way. For example, in the PDS example if no DSLAM can be selected at the one exchange, a new DSLAM can be ordered to fulfill this customer order or it is refused. In this instant, refusing the order might be acceptable but will present a similar problem if similar customer order being placed at the same exchange at a later date. If, however, a new DSLAM is ordered, it may be a waste if future demand is not high enough to utilize this new DSLAM. Thus, in order to handle such request, a careful analyze of the current system needs to be performed.

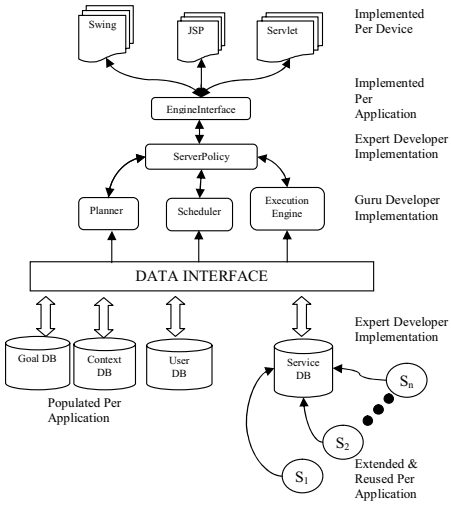

Figure 1: The architecture of a Kreno agent.

3.1 The User Interface

The Kreno user interface is composed of two main component: the developer and the end user components. The former one allows the developers to define web services interfaces as well as specifying the core knowledge base and the abstract targets (goals). The latter component enables end users to select targets that they want Kreno to accomplish, to enter their dynamic constraints and to view the progress of Kreno in handling their requirements.

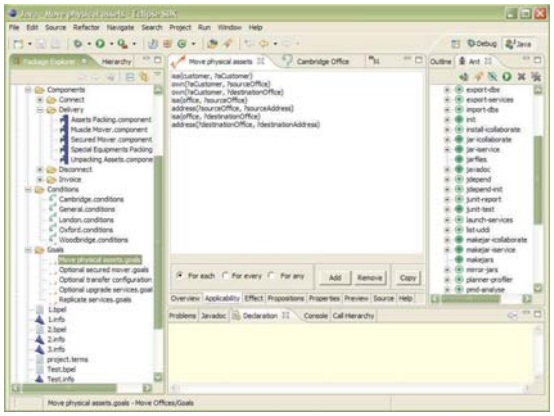

Figure 2: Kreno developer component.

The developer component has been implemented as an Eclipse plugin (see figure 2). Here, the interfaces to various web services can be described as well as the core knowledge base and the abstract goals can be declared.

The end user interface has been implemented as an J2EE - web browser centric system. It allows the users to do a variety of tasks, including:

- make adjustment to the core knowledge base: user can modify/add/remove any proposition or set of propositions if they are no longer valid. Please note that only the core knowledge base can be alter, users have no control over the additional runtime generated knowledge base.
- select targets for Kreno to execute: as mentioned above, the abstract goals of Kreno will be unified against the runtime knowledge base and the final concrete goal set will be presented for users to select.
- enter constraints/requirements: users can add/remove/modify any global constraints/requirements or local to a particular web service.
- view the status of the internal process in Kreno: the results of Kreno will be displayed to end users, including those of planning, scheduling and execution steps.

3.2 The Internal Reasoning And Planing Elements

In order to fulfill customer requests, three sub components of Kreno will be used. First, the planner will be invoked to create a basic plan consisting of a set of web services to be executed in order. Next, the scheduler will be called to allocate time and resources needed for these web services and transform this plan into a concrete workflow. Finally, this workflow will be executed and monitored by the execution engine and any exception will be handled.

Here, the planner performs the service composition via a straightforward means end planning episode based on Graphplan approach [1]. Basically, it constructs an ordered and leveled graph from the relevant and available web services, starting with the propositions from the knowledge base. After each level is constructed, the goal propositions will be checked for validity. If they can be obtained at that particular level, the planning will stop and the graph back tracing is carried out to extract the path to construct the core plan.

If this core plan can be found and extracted from the graph, the scheduler will be responsible for allocating time and resources for each web service in the order of execution to create a detailed workflow. It does so using a look up resource table consisting of time, resources for these web services requirements. If the core plan can be scheduled, the result is a concrete workflow. However, it is possible that some web services cannot be scheduled (e.g. conflict with others, different commitments). If it is the case, the scheduler will try to find alternative time to schedule within the time constraints. Failure to do so will lead to the planner being notified to generate an alternative plan and the scheduler process will restart.

After a concrete workflow has been scheduled, the execution engine will start invoking the web services in the scheduled order. During the execution phase, if an web service could not be invoked for any reason (e.g. unavailable, uncommunicable) or failed to finish, the execution engine will try to invoke a Dynamic Repair System (DSR) to find another compatible web service to carry out the failed task within the required time. Failed to do so will lead the planner to be notified and the whole process will restart.

Having described the technology in the Kreno framework, we present examples of implemented applications in section 4. We also highlight how our approach addresses the business issues around each of these applications.

4. EXAMPLES

This section showcases how Kreno provides an efficient solution to the PDS problem mentioned in section 1 as well as two other examples: a service broker for moving offices and a composer for providing combinations of multimedia services to home users.

4.1 Broadband-Delivered Private Data Service

To handle PDS orders, a number of services have been developed, which include: an ED service to connect to and query the nationwide database of exchanges and DSLAMs, an address finder service to match a given address with the correspondent exchange, a remote data service that allows storage of various types of information, some services to install, upgrade and cancel services with exchanges or DSLAMs.

After an order has been submitted by a customer, the two locations in the order will be queried by the ED service to check for its SDSL capacity (see figure 3). Depending on the result of the ED service, different propositions will be generated (e.g. *nosdsl(locationA)* to indicate that the first location of customer is not SDSL enabled). If the customers locations are SDSL enabled, the correspondent DSLAMs will be analyzed and again, propositions will be generated to indicate the status (e.g. *hasBandwidth(dslam1, locationA)* to indicate that DSLAM dslam1 at locationA has enough bandwidth to carry out the customer request).

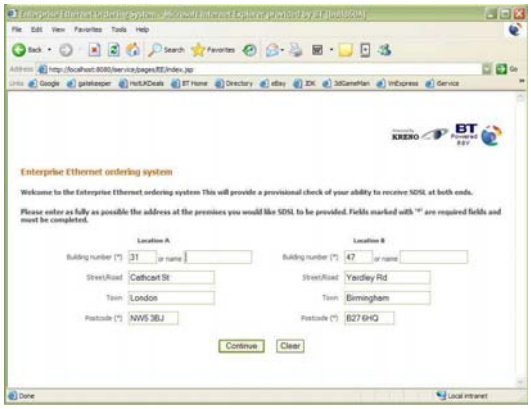

Figure 3: Location input screen.

These generated propositions will control the flow of the plan that Kreno will generate. For example, the install new SDSL capable exchange service can only be executed if the proposition *nosdsl(locationA)* exists. Another example is the actual connection service can only be invoked if all the propositions *hasBandwidth(?dslamA, locationA)*, *hasPorts(?dslamA, locationA)*, *hasBandwidth(?dslamB, locationB)*, *hasPorts(?dslamB, locationB)* exist.

There are situations where no plan can be found. For example, one of the customer's location is not SDSL enabled and it is not yet possible to install a new exchange or the customer has requested the PDS product to be installed on a particular date and no resource can be scheduled. In these cases, Kreno will refuse the orders.

the AI planner in the Kreno agent is used to solve the problem of conditionality and context dependence that has been the source of confusion in PDS orders. Resolving this using traditional programming and K/B technology has proven to be knowledge intensive.

On the other hand, if a plan can be generated, the scheduler component will be responsible for assembly a concrete workflow with the time required for each step. Later, if all the services can be scheduled, the generated workflow will be executed and monitored by Kreno (see figure 4). It is completely automated up to the point where order is generated and passed to field engineers to carry out actual work. Users can track their order via the Kreno web portal and if any exception occurs, it will be handled by the failure

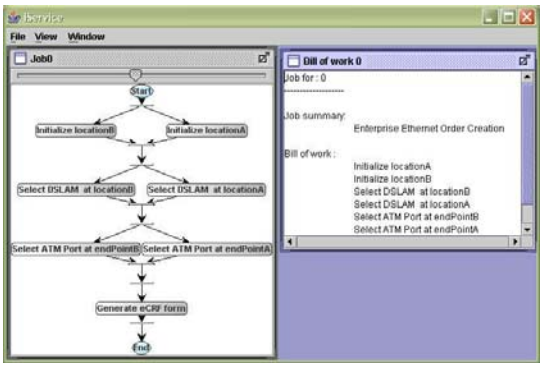

Figure 4: A sample PDS workflow.

management in the Kreno execution engine.

As can be seen, our Kreno framework can be used to provide a practical and automated solution to this particular business problem, which has proven to be time consuming and error prone.

4.2 Moving Offices

Another example in applications we have implemented uses Kreno to orchestrate the moving offices of a typical SME (see section 1). Assuming a SME user has a number of offices scattered around the country and now some of these offices need to be relocated into new premises. This requires a large amount of tasks to be orchestrated concurrently such as: disconnecting currently connected services (e.g. electricity, broadband, etc.) and reconnecting them at the new location, physically moving offices assets, transferring configurations between services, etc.

Figure 5: Moving Offices helper interface.

Kreno helps the user to efficiently organize such jobs by creating a bill of work with the order of the jobs to be executed and can even complete some of the jobs on the users behalf.

At runtime, Kreno allows user to modify the knowledge base to add or remove new offices together with modifying assets and ser-

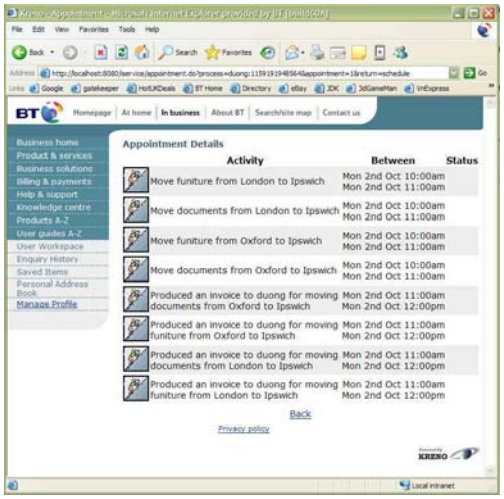

Figure 6: Sample workflow for Moving Offices scenario.

vices in each office. The abstract goals will then be unified with this knowledge base to create a more concrete goal set to be presented to end user (see figure 5). Previously, goals are typically declarative and must be elaborated in advance. Thus, if the environment changes, these goals might not reflect up to date changes and might not be able to capture the desires of the users. By incorporating this new abstract goal mechanism, Kreno can detect and adapt to changes in its working environment and thus, it will help users specifying their requirements correctly.

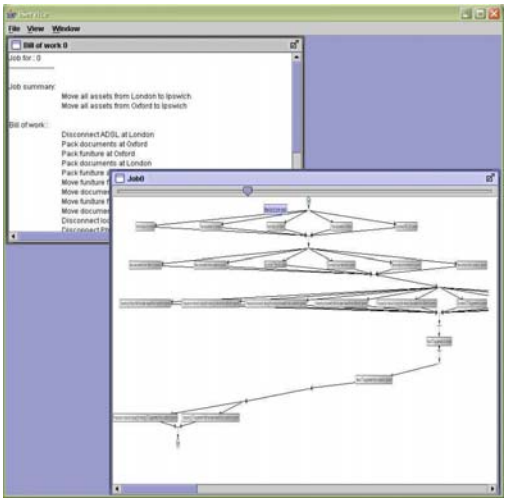

Figure 7: Workflow execution.

After entering constraints for Kreno to process, a basic workflow is generated and presented to the user with the detailed proposal of execution time (see figure 6). This has been worked out first by the planner selecting a subset of the available services to be able to handle the moves and then the scheduler ensuring that these services will be invoked at a particular time to suit the customer's requirements (see section 3.2).

Behind the scenes, the workflow generated will be executed and monitored by Kreno (see figure 7). User can see the results at each step by coming back to the Kreno status page. After all the services complete, the process of moving offices for the customer is also finished. During the execution, Kreno can act as the self service for the users and provide information about its progress. Thus, by providing a scheduling of required tasks and even doing some of them on their behalf, Kreno could help ensuring that the moving schedule will be strictly followed and the required jobs will be completed as planned.

4.3 Entertainment Portal For End User

This application aims to provide network and entertainment services such as tv, movie, gaming to end users. In this complex digital world, there are a plethora options for end users to select. For example, which package of tv they want, how fast their internet connection should be, etc. For a technical savvy user, this should not present any problem but for an ordinary user, appropriate selection is not easy.

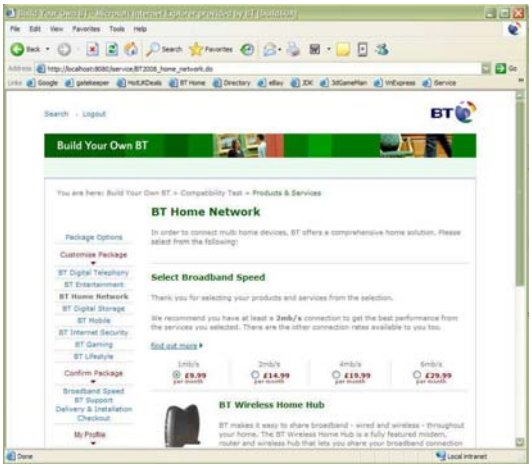

Figure 8: Broadband selection screen.

In addition, there are numerous bundles dealing with various combination of services such as "a discount on broadband service offered when user takes high value movie packages", or "if user takes the fastest broadband package and any expensive tv package then an additional mobile service can be offered for free". Such diversity of selections make the selection of the right combination of packages an even harder task for average users.

To this end, Kreno has been adapted to be used as the composer for helping end users to customize their home network and entertainment services. The idea is that by using a tailored recommendation, end users should get more for their money and the providers will be able to maximize their market profit by providing the right services to the right customer.

Here, end users are presented with a number of choices, including home phone, broadband connection (see figure 8), tv packages, digital storage, gaming connection, etc. These choices are basic and designed to be need based rather than service offering. Based on this information, Kreno will be able to work out the optimum packages by looking through all the combination and select the one

that is most suitable for user requirement.

Specifically, each of the choices presented to the end user is a set of propositions (e.g. {*need(customer, broadband)*, *speed(broadband, 8mbps)*}). These choices are inter-related such as when the user selects premium gaming connection, the broadband connection is automatically upgraded to the highest option (currently at 8mbps). These inter-relations are transparent to the end user and also are represented using different propositions. For example, in order to have broadband, a telephone line must be presented. This is represented in the precondition interface of the installation broadband service in a form of the proposition *has(customer, phone line)*. By embedding such inter-relationships amongst the services, the final plan, once found, will be guaranteed to be valid and will satisfy customer requirements at the minimum cost.

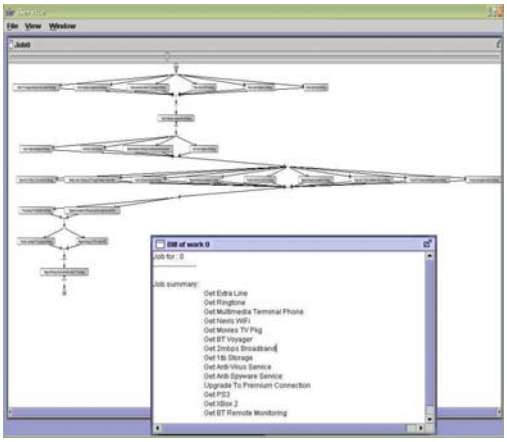

Figure 9: An example workflow generated.

As can be seen, the customer experiences is very straightforward with easy and intuitive interface. All it requires is that customer inputs his/her basic requirements and Kreno will work out the most rational combination for their needs. Figure 9 shows the an example of the generated workflows for a particular customer order.

5. CONCLUSION AND FUTURE WORK

In this paper, we have discussed our Kreno agent framework, which is specialized in handling order entry and management issue in business computing. The benefits as well as the limitations of our framework are recapped at the section 5.1 and 5.2 as follows:

5.1 Application And Architectural Benefits

Kreno platform was designed to be browser-centric and developed based purely on industrial approved standard (see section 3). This allows Kreno to be able to communicate with other industrial applications/platforms and potentially will be able operate in various standard service oriented environments (SOE).

Recap from section 2, various services from different vendors can be generically represented and thus appropriate workflows could be constructed without the need to understand underlying architecture of each and individual service. This allows Kreno to be able to offer user context dependant service selection as well as to efficiently manage the workflows to handle end user's complex orders.

Last but not least, Kreno has the abstract goal representations which allows detecting and reflecting changes in its environment and offering end users tailored goals to correctly capture their requirements.

At the time of writing, Kreno has only been implemented as prototypes (see section 4) and not yet practically deployed. Various efforts is being tried to improve this situation.

5.2 Known Technical Limitations

There are few limitations of the current implementation of Kreno. First, the planning algorithm we used is Graphplan and it is only able to handle mutual exclusion of two items (e.g. service A and service B is mutually exclusive). There are scenarios in which mutual exclusion of more than two items must be detected otherwise the graph building will stop earlier than it should be and even though the goal propositions can be found but no plan could be extracted. It is circumvented at the moment by allowing the graph building to keep repeating the last level until a plan can be extracted. However, this is only a temporary solution and need to be addressed in the future.

Second, Kreno is unable to handle quantification. As a consequence, complicated orders that required specific number of repetitive services could not be handled properly. The solution to this is currently being researched.

6. REFERENCES

[1] A. Blum and M. Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.

[2] R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence Journal*, 2:189–208, 1971.

[3] N. Gibbins, S. Harris, and N. Shadbolt. Agent-based semantic web services. In *Proceedings of the 12th international conference on World Wide Web*, pages 710 – 717, Budapest, Hungary, 2003.

[4] D. Greenwood and M. Calisti. Engineering web service - agent integration. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 2, pages 1918–1925, 2004.

[5] Jason agent framework. http://jason.sourceforge.net/.

[6] N. R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

[7] D. Morley and K. Myers. The SPARK Agent Framework. In *Proceedings of the Third Int. Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS-04)*, pages 712–719, New York, NY, 2004.

[8] J. Rykowski and W. Cellary. Virtual web services: application of software agents to personalization of web services. In *Proceedings of the Sixth International Conference on E-Commerce*, pages 409–418, Delft, Holland, 2004.

[9] S. G. Thompson, M. Cioffi, H. Gharib, N. Giles, Y. Li, and T. D. Nguyen. From Trips to Telcos, Next Generation Service Portals. *BT Technology Journal*, 24(1):27–39, 2006.

[10] S. G. Thompson, H. Gharib, N. Giles, Y. Li, and T. D. Nguyen. Using ai and semantic web technologies to attack process complexity in open systems. *Knowledge Based Systems*, page to appear, 2006.

[11] W3C Standards. http://www.w3.org/.

[12] WS-BPEL. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/.