A Distributed Workflow System with Autonomous Components

Maryam Purvis, Martin Purvis, Azhar Haidar, and Bastin Tony Roy Savarimuthu

Information Science Department, University of Otago, Dunedin, New Zealand
{tehrany,mpurvis,tonyr}@infoscience.otago.ac.nz

Abstract. This paper describes the architecture of a distributed workflow management system in a dynamic environment. The system features autonomous agent components that can adapt to both structural changes in business processes and changes in system parameters, such as the number of available resources. This adaptation could be a permanent adjustment that should be reflected in all the incoming work cases, or be associated with a particular instance of a work case. In addition, parts of the system can be modified by observing the behaviour of the system for possible shortcomings due to a non-optimal distribution of resources or faulty inter-process dependencies which could result in bottlenecks. Because of the autonomous nature of subsystem components, the workflow system can adapt to changes without the necessity of centralized control. The architecture of the system is described in the context of a distributed workflow example.

Keywords: dynamic workflow, autonomous components, interaction protocols, coloured Petri nets, adaptability

1 Introduction

Workflow management systems (WfMS) [1-3] are increasingly being used to manage business processes associated with distributed global enterprises. Some of the benefits of using a WfMS are

- ability to visualize the overall process and interdependencies between various tasks,
- automation of the processes, and
- coordination and collaboration between various business entities.

Traditionally, however, most WfMSs have had centralized control architectures along with fixed process model specifications. The current research trend is in the direction of (a) more distributed architectures which can reduce potential bottlenecks with respect to particular system components and (b) more flexible process model specifications, which can accommodate dynamic and changing requirements that occur in today's business environment [4,5].

It is often desirable to have the capability of modifying the existing process model due to changing external influences or of dealing with exceptional cases in which the normal processes may not be appropriate. In the past WfMSs were used in well-defined activities, such as manufacturing, where the processes tend to be more established and stable. But in the current climate WfMS may be used in connection with more fluid business processes, such as e-commerce, or in more complicated processes

M.W. Barley and N. Kasabov (Eds.): PRIMA 2004, LNAI 3371, pp. 193–205, 2005.

involving human interactions, such as the software development process. In such situations, at times, it is not always possible to predict in advance all parameters that may be important for the overall processes. In addition, it is often appropriate for certain groups within a distributed organisation to be autonomous and not always under centralized control. Consequently it would be helpful if we could design WfMS systems that could cope with these dynamic requirements and provide some level of process modification. It is important to make the workflow system dynamic and adaptable, since workflows of multi national companies span across countries. For example the main workflow might be present in New Zealand and the sub processes could be distributed in countries like India and Germany.

One of the benefits of using a WfMS is to be able to streamline processes associated with an organization and be able to visualize some of the interdependencies between various tasks or various processes in a larger context. It is desirable to represent these processes in a formal way that could be used for further analysis and at the same time have a graphical and intuitive representation. The coloured Petri net (CPN) notation [6] meets this requirement. In the past, the CPN formalism has been used successfully to model the dynamic behaviour associated with particular processes representing various activities of a complex system, such as business processes. In the context of the WfMS, CPNs have been used to specify the process model of a WfMS component [2,7], and CPNs have been used to model processes generally, since they offer a well-established modelling technique that combines expressiveness, simplicity and formal semantics. However, in the present work we are extending this idea so that the various sub-processes associated with a large enterprise could be distributed on different hosts, while at the same time being interconnected with one another according to the overall process model associated with a given organization.

An advantage of having a formal representation that is executable is that one can examine the behaviour of the system according to various what-if scenarios that may be considered as a result of potential changes to the process or some of the model parameters such as the various constraints that might affect the outcome. By simulating the model for typical scenarios, it is possible to analyse the outcome of the simulation and fine-tune the specified resources or constraints so that more favourable results can be achieved; and this is also possible with coloured Petri nets.

2 Architecture of the System

To accommodate this level of adaptability, the system should be flexible and made of loosely coupled modules. Our workflow system uses JFern [8], a Java-based tool for the enactment and simulation of coloured Petri nets. We are also using the Opal agent framework [9], which conforms to the Foundation for Intelligent Physical Agents (FIPA) specifications [10] and which provides an agent-based infrastructure for the support of distributed, adaptable computing.

The system architecture (shown in Figure 1) is based on a framework that was developed by the NZDIS research group [11]. In this framework various agents are responsible for performing their tasks by executing a model of their activity specified with Petri nets. The open and dynamic nature of the agents facilitate the incorporation of adaptable process models. Each model is associated with a sub-process associated with the overall workflow.

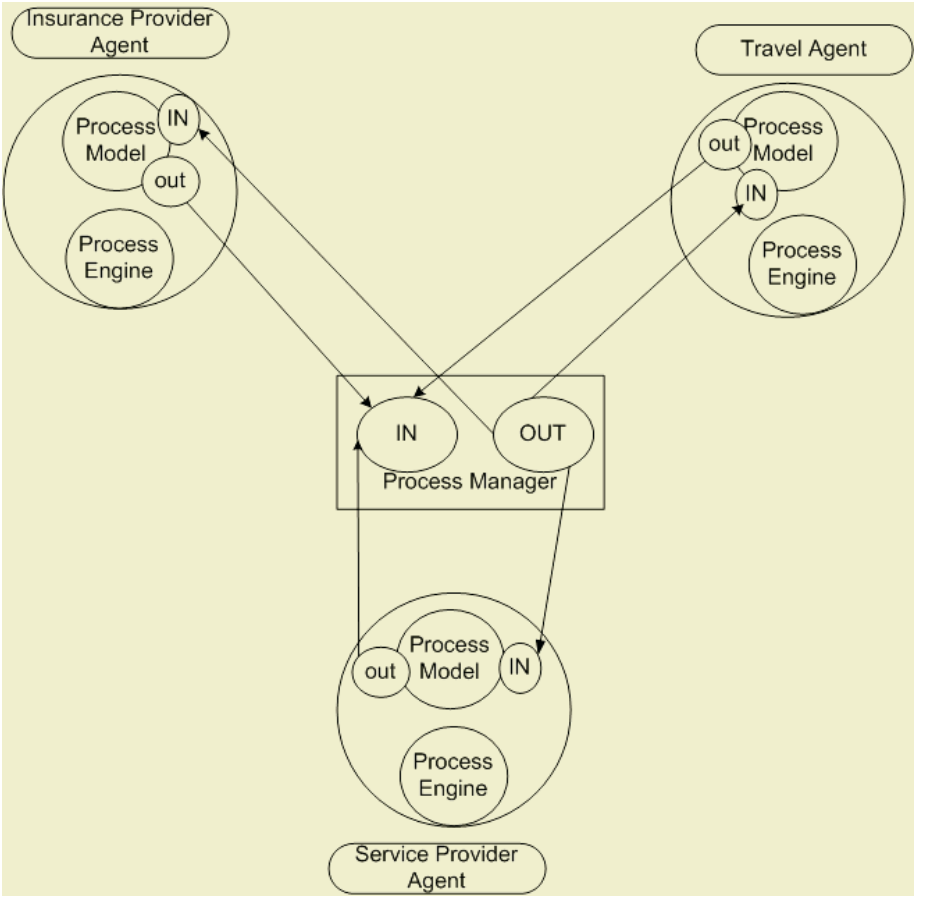

Fig. 1. Architecture of the agent system.

Each agent runs an instance of JFern for Petri net protocol enactment. The agents interact by sending messages to other agents as specified in their protocol model. When an agent receives a message, the appropriate information is deposited in an *In* place in its Petri net, and this may enable transitions to be fired that are associated with the protocol model. Similarly any message going to another agent is deposited in the *Out* place. All these nodes are fused with the *out* place of the process manager. The process manager dispatches the messages to the appropriate agents as specified in the message content.

An agent can receive a proposal for a new or modified interaction protocol, associated with its participation in the overall workflow, from another agent by means of a FIPA-specified *propose* message. The content of this message contains the proposed interaction protocol encoded in XML format. The interaction protocol actually comprises a coloured Petri net and the associated ontology, which describes the terms used in the model and their relationships. The ontology is represented in UML, and both the Petri net and the UML-encoded ontology information are encoded in XML and sent together as the overall interaction protocol. Because the agents are autono-

mous, they may not agree to the new proposed protocol and may inform the proposing agent of their refusal to agree. Under certain circumstances, such as in loosely-organised confederations of service providers that are distributed across the Internet, this option of refusing the newly proposed protocol may be appropriate. The system architecture described here provides support for this kind of semi-autonomous workflow structure.

The agent-based architecture also supports the notion of incorporating new agents appearing on the scene (joining the agent group) and offering new services on the fly. These new agents will be informed on arrival of current interaction protocols for the group by means of the same *propose* message mentioned above.

The governance of the interaction protocols is handled by one or more 'manager' agents, which maintain a model repository. At the present stage of technology, such manager agents are expected to be interfaces to human managers. Thus if it is determined during the middle of workflow execution that a new model is required, the manager or workflow designer would have the opportunity to create a new model and register it with the 'manager' agent's model repository which can then be distributed to the appropriate agent that may require an alternative protocol.

A separate workflow designer component can exist on different hosts. The workflow administrator of a branch of an organisation can design the process associated with that particular office and send the model and the associated work cases to a specific agent.

The system architecture comprises several components including the workflow engine, workflow modeller, and various services such as an XML-to-Petri net (in Java) translator, and generic service provider agents that can locate a resource and provide a service for a particular task.

This architecture allows for monitoring of the system based on a set of predefined conditions such as availability of resources, which could be used as a feedback mechanism for human administrators.

2.1 The Workflow Modeller

This workflow modeller component is used to specify the processes associated with performing a particular activity. Coloured Petri nets are used to model workflow systems, due in part to their sound mathematical foundation and to the fact that they have been used extensively for modelling of distributed systems [12]. Coloured Petri nets consist of the following basic elements:

tokens which are typed markers with values - the type can be any Java class.
places (circles), which are typed locations that can contain zero or more tokens.
transitions (squares), which represent actions whose occurrence (firing) can change the number and/or value of tokens in one or more of the places connected to them. Tokens may have guards which must evaluate to TRUE in order for the transition to fire. In a workflow model a transition may represent a task.
arcs (arrows) connecting places and transitions. An arc can have associated inscriptions, which are Java expressions whose evaluation to token values affects the enabling and firing of transitions.

Some reasons for preferring Petri net modelling in connection with workflow modelling to other notations are:

They have *formal semantics*, which make the execution and simulation of Petri net models unambiguous.
It can be shown that Petri nets can be used to model workflow primitives identified by the Workflow Management Coalition (WfMC) [13]
Typical process modelling notations, such as dataflow diagrams, are event-based, but *Petri nets can model both states and events*.
There are *many analysis techniques* associated with Petri nets, which make it possible to identify 'dangling' tasks, deadlocks, and safety issues.

Other standardization protocols do not cater to expressiveness, simplicity and formal semantics. The comparison of high-level Petri nets with other proposed standardization protocols can be found in [15].

The Petri net models created using the JFern engine are instantiated as workflow components in our system.

2.2 Workflow Engine

The workflow engine is the component that executes the interaction protocol that has been modelled using Petri nets. The JFern tool can be used as a process modeller and also the execution engine.

2.3 Conversation Manager

The conversation manager is the component that organizes the interaction between various interaction protocols. It is responsible for dispatching the messages from the "in" and "out" places and the ontology component which defines the terms that appear in the model (the places, transitions and the arc expressions). The conversation manager plays the role of the resource manager. It identifies the list of resources that can perform a certain task. These resources can be chosen from a pool of resources available in the form of 'agent societies' [17]. The conversation manager can choose a service provider agent that is less flexible and less expensive than some other provider that offers more expensive services. Each agent in the society has certain capabilities inscribed as attributes.

3 Example Scenario

In order to show the operational aspect of the system, as well as how it can adapt to changes, an example scenario is described. In this scenario, various sub-nets associated with different sub-processes of the system are discussed. This model has been adapted from a travel agent model example discussed by Van der Aalst [2].

3.1 A Distributed Process Model

In this scenario the interactions involving a customer, a travel agent, a transport ticket seller (travel service provider) are described. Figure 2 depicts a simplified version of the interaction protocol for the travel agent. The protocol is initiated when a customer's request has been submitted to the travel agent, indicated in the model by the placement of a token at the *In* place of the net. The travel agent then searches some

external database (not shown in the diagram) to come up with some possible trip options (the *Prod Opts* transition). The result of the search is placed in the *Opts* place. These options are then placed in the *Out* place so that they can be sent back to the customer. At this point the customer is contacted (the customer interaction is not shown in this diagram). When the customer responds, the travel agent's *Get Cus Res* transition will fire. Either the customer will select an option for purchasing a ticket (an external travel service provider will have to be contacted for the purchase of such a ticket) or the customer will not be satisfied with the options he was sent and will need more options (*Need More Opts*). Assuming that the customer does select one of the options for purchase (as indicated by the value of the token in the *Cus Res* place), the *Res Tick* transition is enabled, causing the travel agent to send a ticket reservation request to a travel service provider, such as a bus company or sightseeing operation. A copy of the customer's ticket reservation request is kept in the *Res Sent* place for later consultation. The travel service provider will either send back a notification that a reservation has been made (enabling the *Get Tick Res* transition) or send back notification that there are no tickets available (enabling the *Get Rej* transition, which will cause a notification of that fact to be sent back to the customer). If the travel service provider *does* return a confirmed ticket reservation, it is matched with the ticket reservation request stored in the *Res Sent* place and then deposited in the *Tick Res* place. This will, in turn, enable the *Send Bill* transition, causing a bill to be sent to the customer for payment. After payment is received, the travel agent will send the payment to the service provider, get the ticket from the service provider, and then forward the ticket on to the customer.

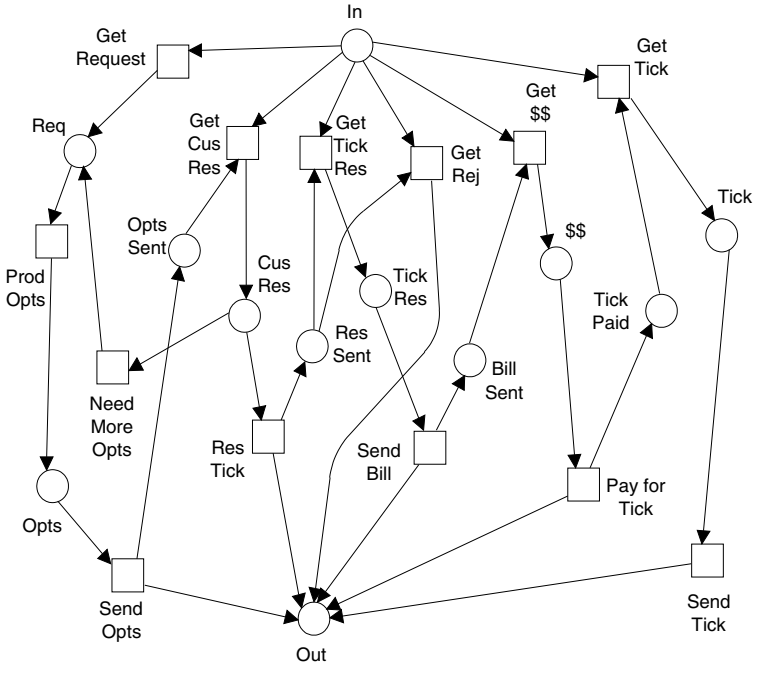

Fig. 2. Interaction protocol for the travel agent.

Note that information is stored in the *Opts Sent*, *Res Sent, Bill Sent, and Tick Paid* places for matching up with later messages that arrive. This enables the travel agent to conduct activities with many customers and travel service providers concurrently.

Figure 3 shows the interaction protocol[1] for the customer. This protocol has a *Start* place that has a token placed in it (specifying the customer's travel interests) when the customer wants to initiate a conversation with the travel agent. The *Send Request* transition causes the request to be placed in the *Out* place for sending a message to the travel agent and a copy of the request is stored in the *Req Sent* place. Later, the customer expects to receive a set of options for selection from the travel agent, and these options should match his or her travel request. After an option is selected, this is placed in the *Out* place for sending back to the travel agent, and a copy of the reservation selected is stored in the *Res Sent* place. Subsequently, the customer expects to get a bill, pay it, and ultimately get tickets matching what he or she has paid for.

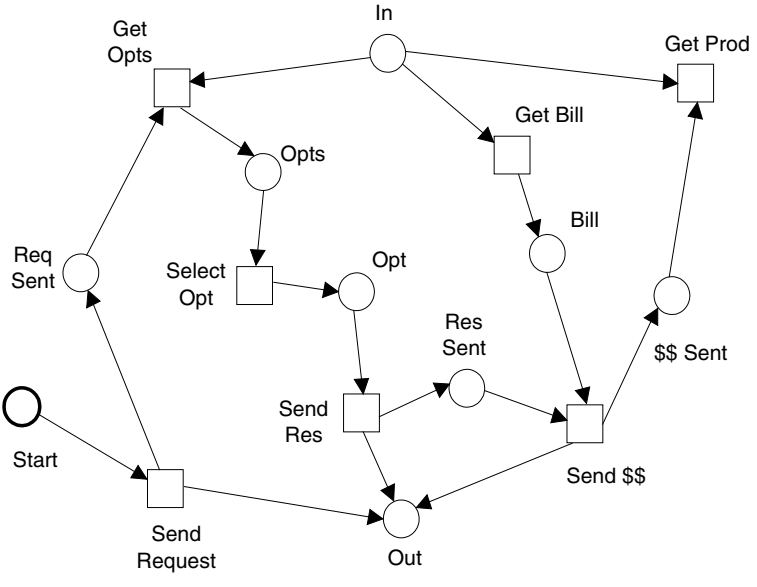

Fig. 3. Interaction protocol for the Customer.

Figure 4 shows the interaction protocol for the travel service provider. The travel service provider might supply any kind of travel service, such as boat passage, tramping guides, etc. The travel service provider initially receives a message from the travel agent indicating that a reservation has been requested for his or her service, such as a transport ticket. The service provider must then see if the requested resource (usually a ticket booking) is available. So both the *Prep Prod* and *Send Reject* transitions examine the single token located in the *Available Resources* place. The single token in the *Available Resources* place contains a list of available resources, and information

[1] At times we use the term protocol to refer to the activities of individual participants and at other times to the collection of activities of all participants. The context should make clear the difference.

for the list in this token is maintained by access to an external database. The *Prep Prod* transition is enabled if the relevant information (*i.e.* what is desired, for example, a bus ticket) on the reservation request token in the *Res* place matches up with one of the resources listed on the token in the *Available Resources* place. On the other hand, the *Send Reject* transition is enabled if the information on the token in the *Res* place fails to match up with an item listed on the token in the *Available Resources* place. In the case where there are tickets available, the service provider then prepares the product (a ticket, say) and sends the bill back to the travel agent and keeps a copy of it in the *Bill Sent* place. When payment is received later, the service provider will send the product that has been stored in *Prod Ready*. In the simplified scenario described here, there is only a single generic protocol for a travel service provider shown, but there could be many such protocols that are used for particular service providers. There could also be more complicated interactions with the customer. In our example, payment is made directly to the travel agent. But there could be other options available, including having the travel agent act as a broker, with payment transactions ultimately taking place directly between the customer and the travel service provider.

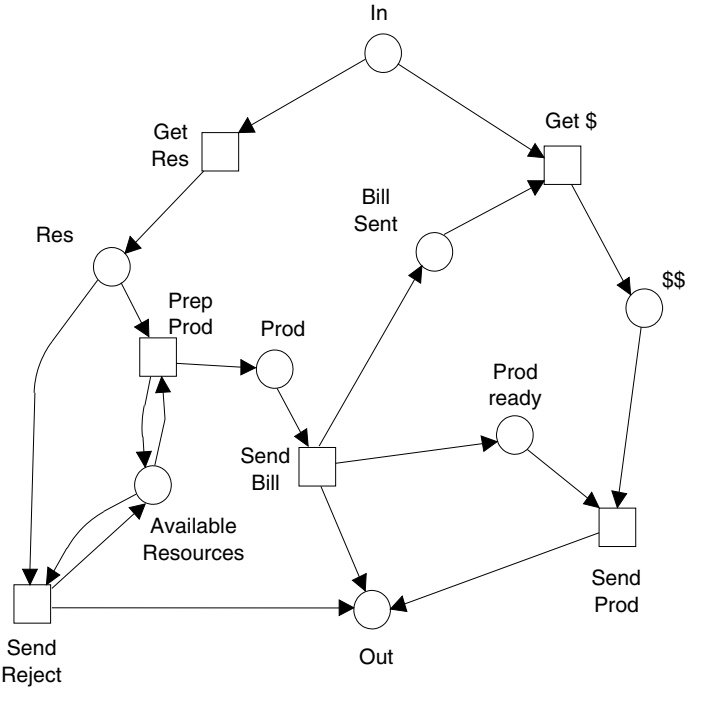

Fig. 4. Interaction protocol for the service provider.

Figure 5 shows how all the interaction protocols are created and executed. The human manager can create interaction protocols using the JFern process modeller and store them in the system using the storage agent. The stored protocols can be viewed querying the storage agent. When sets of interaction protocols are to be executed, the protocols are selected and submitted to the workflow engine.

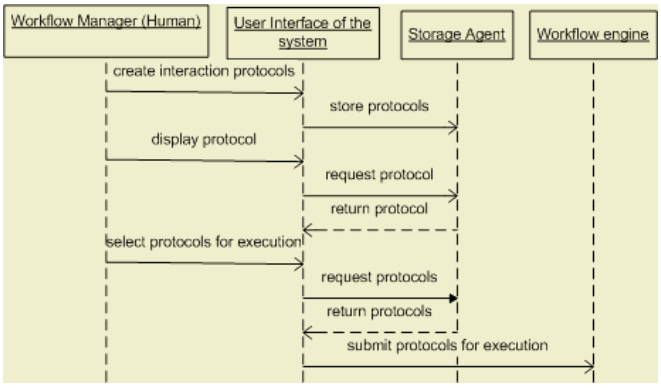

Fig. 5. Creation and execution of the interaction protocols.

The conversation manager plays the important role of the co-ordinating agent between various interaction protocols. It is responsible for matching the "in" and "out" tokens from various agents. Figure 6 shows the interaction between a customer agent, travel agent, service provider and the conversation manager.

The customer agent executes the interaction protocol and places a request for reserving a ticket to the travel agent through the conversation manager. The conversation manager transfers the requests to the travel agent from the customer agent. This corresponds to the transfer of a token from the *out* place of the customer agent to the *in* place of conversation manager. The token is then placed at the *out* place of the conversation manager which is appropriately moved to the travel agent's *in* place by the conversation manager.

The travel agent could then get the appropriate service provider to perform a particular task from the resource agent. All these interactions are co-ordinated through the conversation manager.

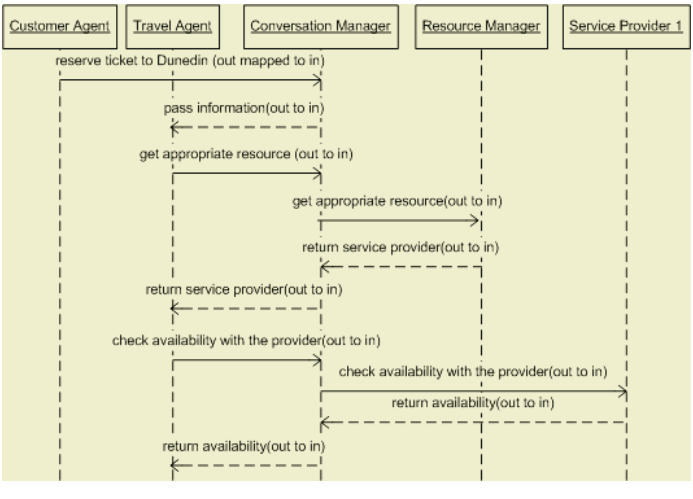

Fig. 6. Interaction protocol for the service provider.

3.2 Adaptive Workflow Process Operation

Consider now an international travel agency with individual travel agents spread across the globe (or region). The individual agents may be using an interaction protocol associated with customers and service providers such as we have described in Figures 2-4. These sets of interaction protocols represent the workflow cases for the travel agents of the agency. Suppose, now, that a health crisis emerges in some regions of the world, and that the global manager of the travel agency decides to recommend a new interaction protocol for some of his or her travel agents.

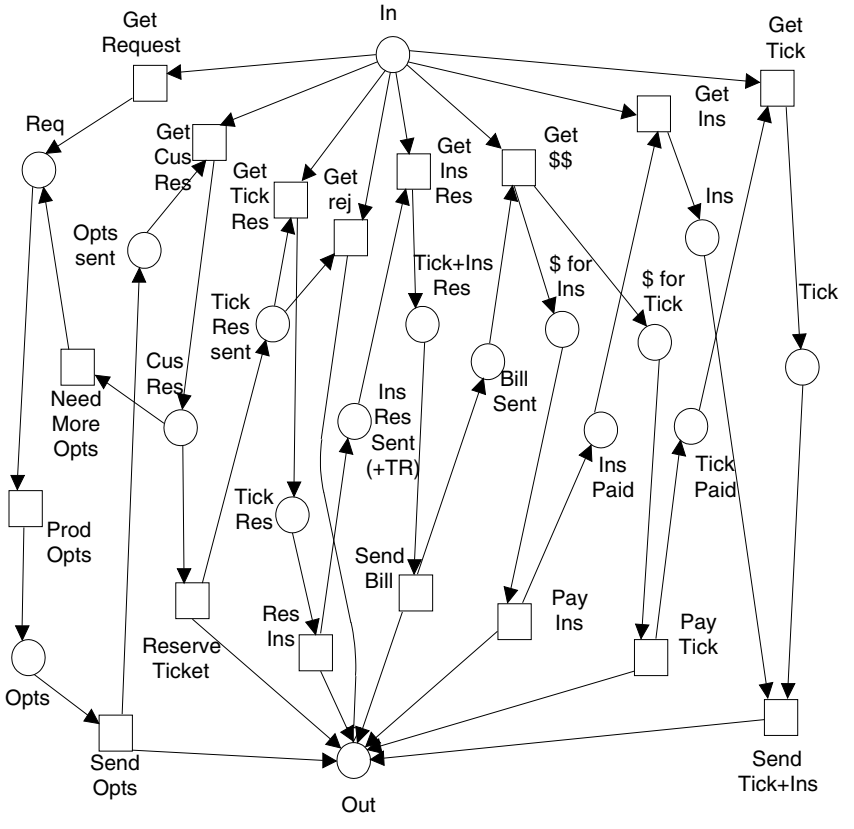

Fig. 7. New interaction protocol for travel agent involving two coupled service providers (for tickets and insurance).

The newly proposed protocol is to require that all ticket transactions must be bundled with a health insurance policy that is offered by some recommended health insurance agents. This new interaction protocol is now recommended for those travel agents in parts of the world that are affected by the health crisis, and the new travel agent protocol is shown in Figure 7. The entire protocol is sent to all travel agents in the organisation in the form of an encoded XML expression in the body of a FIPA *propose* message. Those travel agents that are dealing with customers in affected

areas would be urged to adopt the new protocol. For a resilient and adaptive global organisation, this kind of autonomy may be essential for success in a competitive environment.

In the new protocol, there is now a ticket selling travel service provider and an insurance service provider. For this new scenario, we assume that the customer and both service provider protocols remain as shown in Figures 3 and 4, respectively. Both of the service provider agents use the interaction protocol depicted in Figure 4: they prepare a product when requested by the travel agent, and that product is delivered to the travel agent when payment is received. The protocol for the travel agent is modified, though, as shown in Figure 7. When the initial request comes in from the customer, the early stages of interaction are as before in Figure 2. However after the ticket reservation request is confirmed by receipt of a message from the ticket selling service provider, the travel agent proceeds to request purchase of insurance from an insurance provider (a message to the insurance provider is prepared in connection with the *Res Ins* transaction, and a token for the message is placed in the *Out* place). Information about the insurance request and the confirmed ticket reservation is stored in the *Ins Res Sent (+TR)* place. Later when the bill is sent to the customer and payment is received, the travel agent arranges to pay both the ticket selling service provider and the insurance provider. After the travel agent receives authorisation from both the ticket selling agent (in the form of tickets) and the insurance provider (possibly just some authorisation number) these vouchers are bundled together and forwarded on to the customer.

4 Discussions and Future Work

The ability to design and update interaction protocols that, together, represent workflow scenarios enables an organisation of semi-autonomous entities or agents to respond and adapt to changing conditions in a distributed environment. For illustrative purposes, we have described a distributed example involving travel agents. This is an appropriate example, because the conditions and available service providers are constantly changing in the travel and tourism industry, and it can be difficult to maintain an organised sense of workflow activities under these conditions. As new types of service providers become available, there can be new types of interaction protocols that are appropriate for those service providers, and all the agents that interact with them would need to be informed about those interactions protocols.

Another application domain can be in the area of distributed software development, where many independent, autonomous software developers are working together on a large, possibly open-source, development project. Integration, testing, and acceptance activities can be adapted to deal with changing scheduling requirements, customer-imposed constraints, or preferences among the distributed collection of team members.

This work is also applicable in those areas that are less human-dominated and in which electronic agents are performing most of the work. In these environments, it is essential to be able to monitor and coordinate the activities of groups of autonomous agents. Facilities such as those we are developing can offer more choice in the organisation of distributed enterprises, because they can provide coordination facilities while, at the same time, allowing individual entities to retain more autonomy.

The following enhancements to the existing system are planned for future work in this research.

Provide more explicit facilities for resource management so that conventional workflow models can be incorporated.
Provide a direct interface to one of the exiting analysis tools so that process models can be analysed on the spot. The resulting analysis can lead to improved system performance.
Improve the monitoring capability so that various performance statistics and throughput information is available graphically.
Improve the visualization of linked and hierarchical models.
We are in the process of extending the proposed prototype and evaluating various process model scenarios. In particular we are examining the integration of the web services as discussed by Paul et al [16].

The authors would like to acknowledge the technical support and consultation provided by Mariusz Nowostawski and Peter Hwang of the University of Otago.

References

1. Schael, T.: Workflow Management Systems for Process Organisations. Springer-Verlag. (1998)
2. Van der Aalst, W., Van Hee, K., Schmidt, J. W.: Workflow Management: Models, Methods, and Systems. MIT Press. (2002)
3. Meilin, S., Guangxin, Y. , Yong, X. , Shangguang, W.: Workflow Management Systems: A Survey. In: Proceedings of IEEE Intl. Conf. On Communication Technology, Beijing, (1998)
4. Borghoff, U.M. , Bottoni, P. , Mussio, P., Pareschi, R.: Reflective Agents for Adaptive Workflows. In: Proc. 2nd Conf. on the Practical Application of Intelligent Agents and Multi_Agent Technology (PAAM'97), London, U.K., (1997) 405-420
5. Stormer, H.:A Flexible Agent-Based Workflow Systems. In: Workshop on Agent-Based Approaches to B2B, Fifth International Conference on Autonomous Agents, Montreal, Canada (2001)
6. Jensen, K.: Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use, Vol. 1: Basic Concepts. Springer-Verlag, Berlin (1992).
7. van der Aalst, W.M.P.: The application of Petri nets to workflow management. In: The Journal of Circuits, Systems and Computers vol. (1998) 8(1), 21-66.
8. Nowostawski, M.: JFern, version 1.2.1, http://sourceforge.net/project/showfiles.php?group_id=16338 (2002).
9. Purvis, M., Cranefield, S., Nowostawski, M., and Carter, D.: Opal: A Multi-Level Infrastructure for Agent-Oriented Software Development. In: *Information Science Discussion Paper Series*, No. 2002/01, ISSN 1172-6024, University of Otago, Dunedin, New Zealand.
10. FIPA. Foundation For Intelligent Physical Agents (FIPA). FIPA 2001 specifications, http://www.fipa.org/specifications/ (2003)
11. Purvis, M. K., Huang, P., Purvis, M. A., Cranefield, S. J., and Schievink, M.: Interaction Protocols for a Network of Environmental Problem Solver. In: Proceedings of the 2002 iEMSs International Meeting: Integrated Assessment and Decision Support (iEMSs 2002), Volume 3, Andrea E. Rizzoli and Anthony J. Jakeman (eds.), The International Environmental Modelling and Software Society, Lugano, Switzerland (2002) 318-323

12. van der Aalst, W.M.P.: Three good reasons for using a Petri-net-based workflow management system. In: Navathe, S., Wakayama, T. (eds.): Proc of International Working Conference on Information and Process Integration in Enterprises (IPIC'96),. Massachusetts Institute of Technology, Cambridge, Massachusetts, (1996) 179-201.
13. Workflow Management Coalition: The Workflow Reference Model, Document No. TC00-1003, Issue 1.1. (1995)
14. Theoretical Foundations Group and Distributed Systems Group of the Department of Informatics, University of Hamburg. Renew – The Reference Net Workshop, Release 1.2, (2000)
15. van der Aalst, W.M.P.: Don't go with the flow: Web Services composition standards exposed, Jan/Feb 2003 issue of IEEE intelligent systems.
16. Paul Buhler and José M. Vidal. Enacting BPEL4WS specified workflows with multiagent systems. In *Proceedings of the Workshop on Web Services and Agent-Based Engineering*, 2004.
17. B.T.R Savarimuthu and M.Purvis, A Collaborative mulit-agent based workflow system. In: M. G. Negoita, R. J. Howlett, L. C. Jain (eds.), Knowledge-Based Intelligent Information and Engineering Systems, 8th International Conference, KES2004, Wellington, New Zealand, September 2004, Proceedings, Part II, Springer LNAI 3214, pp. 1187-1193, 2004.