A Framework for Distributed Workflow Systems

Martin Purvis Maryam Purvis Selena Lemalu
Department of Information Science
University of Otago
Dunedin, New Zealand
{mpurvis, tehrany, slemalu}@infoscience.otago.ac.nz

Abstract

Workflow management systems (WFMS) are being adopted to assist the automation of business processes that involve the exchange of information. As a result of developments in distributed information system technology, it is now possible to extend the WFMS idea to wider spheres of activity in the industrial and commercial world and thereby to encompass the increasingly sprawling nature of modern organisations. This paper describes a framework under development that employs such technology so that software tools and processes may interoperate in a distributed and dynamic environment. The framework employs Petri nets to model the interaction between various sub-processes. CORBA technology is used to enable different participants who are physically disparate to monitor activity in and make resource-level adaptations to their particular subnet.

Keywords

Distributed systems, workflow, process modelling, Petri nets

1. Introduction

Developments in networking and telecommunications have in the past few years opened up enormous opportunities for linking up disparate information sources and computational modules. This has raised significant hopes for improvements in the area of software interoperability: the linking of software modules to carry out complex computational tasks. An example of an application that facilitates software interoperability is a Workflow Management System (WFMS), which provides support for the automation of business or industrial processes involving human and machine-based activities. By using such systems, organizations can accelerate throughput, reduce costs, and monitor performance of common, well-understood operational processes in their domain.

Existing WFMSs, however, are practical for only the most straightforward operational processes and are not necessarily suited for the current business climate. This is due to the fact that the current changing, globally competitive business and engineering environment has led in recent years to fundamental changes in the ways organizations themselves are managed and structured. The climate is increasingly marked these days by the need for rapid response to changing consumer demands, intense competition, and a rapidly evolving technical infrastructure [5]. Organizations that once were vertically arranged in hierarchical, bureaucratic structures are becoming increasingly decentralised into geographically distributed semi-autonomous business units so that they can respond more quickly and efficiently to changing market conditions. At the same time, business processes have become more complex, involving the concurrent participation of multiple and distributed functional units. These changing conditions have led to the need for improved workflow management tools that are (a) adaptive to changing conditions and (b) provide assistance in the area of horizontal, cross-organizational management. The focus of this paper is on the development of such workflow management tools.

We note in general that the management of a business process has three basic stages: (1) design and creation, (2) the provision of resources, and (3) enactment. Of course there are great differences in the range of business processes, and they might be loosely scaled according to the following categories:

- *Administrative* – repetitive, predictable processes with simple coordination rules
- *Ad-hoc* – processes that involve more human judgement, such as a sales process
- *Collaborative* – processes, such as system design, that are even less structured and require support for group work.

Existing WFMSs have primarily concentrated on *administrative* workflows, for which the processes are well-understood, and so these systems typically only offer support for *enactment* [11]. As a consequence, such systems often lack support for both design and for

adaptation to the dynamic changes of resource needs and availability. Furthermore, most existing systems adopt a centralised architecture and do not operate across multiple platforms.

The focus of the current work is on the development and characterisation of dynamic workflows (ad-hoc and collaborative processes) in a distributed environment. We contend that the development of distributed, dynamic workflow systems, by facilitating the interoperation of tools and mechanisms in a heterogeneous environment, will be a key factor in the evolving information-intensive economic environment. Consequently they deserve significant attention from the software engineering community.

In this paper the following sections describe:

- the software architecture of the system
- key technical components that are used in connection with this architecture
- an example workflow, taken from the area of university enrolment
- plans for monitoring and adaptive workflow modelling, and
- conclusions and future work.

2. System architecture

The Workflow Management Coalition's (WfMC) Workflow Reference Model (Figure 1) shows the major components and interfaces within the architecture of a WFMS [15], where a *workflow* is defined to be "the computerised facilitation or automation of a business process, in whole or part." The *Process Definition* component is used by the user to define and specify the sequence of operations of a workflow. Typically this will involve the use of some visual modelling representation. The *Workflow Engine* is responsible for the execution of the overall workflow, at various stages of which individual workflow applications (*Client Applications* or *Invoked Applications*) will be accessed or executed.

Although the WfMC has defined a graphical workflow modelling notation as a proposed standard, we use Petri nets to model workflow systems, due in part to their sound mathematical foundation and the fact that they have been used extensively in the modelling of distributed systems [13, 14].

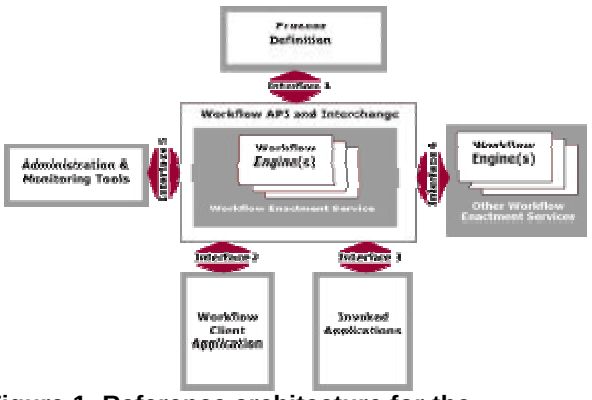

Figure 1. Reference architecture for the Workflow Management Coalition

We use the Renew (Reference Net Workshop) [8] net editor to represent (*Process Definition* component) and execute (*Workflow Engine* component) our workflow models. Renew is written in Java and comes with the complete source code, which has enabled us to make modifications in order to incorporate it into our system.

Renew reference nets are closely related to coloured Petri nets [7] in that they consist of the following basic building blocks:

- *places* (circles) which are typed locations that can contain zero or more tokens – (Renew also has an untyped formalism which is useful for prototyping). An *input place* is connected to a transition by an arc that points from a place to a transition and is examined to see whether or not a transition is enabled. An *output place* is a place to which tokens may be added when a transition fires.
- *transitions* (squares) which are actions whose occurrence (firing) can change the number and/or value of tokens in one or more of the places connected to them. In a workflow model a transition may represent a task.
- *arcs* (arrows) which are connections between places and transitions. Renew has arc types in addition to input and output arcs *e.g. test arcs* (no arrows) that provide read-only access to a token and *flexible arcs* (double-headed arrows) that allow multiple tokens to be moved by a single arc.
- all of the above may have *inscriptions* associated with them (in Renew, inscriptions are Java expressions *e.g.* 'action inscriptions' are a special transition inscription which are guaranteed to be evaluated exactly once during the firing of a transition and can be used to produce side effects).
- *tokens* which are typed markers with values – the type can be any Java class, and, in addition, a simple black untyped token is denoted by [].

Figure 2 shows the use of Java expressions within action inscriptions on the first transition to get an array of tokens representing customer requests and the use of a 'flexible arc' (a special Renew construct) for splitting that array into its constituent elements. The *Order Entry* transition only needs one token in its input place for it to fire resulting in identical tokens being deposited in each of its output places. Once both the *Inventory Check* and *Credit Check* transitions have fired and tokens have appeared in their output places, the *Evaluation* transition may fire. The sequence from the input place of the *Order Entry* transition to the output place of the *Evaluation* transition is the Petri net equivalent of the standard parallel routing construct. When the *Evaluation* transition fires, the token deposited in the output place will reflect the decision and determine whether the *Approval* transition or the *Rejection Letter* transition will be fired. This is the standard conditional routing construct. The model execution finishes when all requests are located in the output place of the *Archiving* transition. This example will be used again in Section 4.

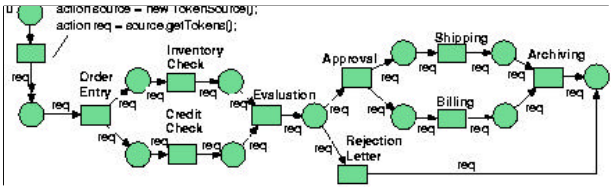

Figure 2. Order processing example

Significant reasons for preferring Petri net modelling in connection with workflow modelling over other notations are [14]:

- Despite their graphical nature, coloured Petri nets have a formal semantics, which makes 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 WfMC [14]
- Typical process modelling notations, such as dataflow diagrams, are event-based, but Petri nets can model both states and events. This enables a clear distinction between task enablement and execution and makes it easier to represent the time of task execution in the model
- The many analysis techniques associated with Petri nets make it possible to identify 'dangling' tasks, deadlocks, and safety issues.

2.1. Workflow engine component

When a Renew net is created using the *Process Definition* component, a static (Java class) structure is created. When a simulation is started by the *Workflow Engine*, a new instance of that net is created. Two such instances can communicate with each other by means of *synchronous channels* [2,8,9,10], which provide a mechanism for the synchronization of two transitions (in two separate nets) which fire atomically at the same time. The initiation ("calling") transition has a special net inscription, called a "downlink", which passes parameterised information to the designated subordinate net/subnet transition. The downlink expression must make an explicit reference to the other net instance, so it takes the form, *netexpr:channelname(arg, arg, …)*. The designated transition in the subnet has an "uplink" inscription, which is used to serve requests from downlink calls. The uplink expression has the form *:channelname(arg, arg, …)*. Hierarchical Petri nets are (usually) those for which a single transition can be substituted by an appropriately structured subnet. Although the reference nets of Renew are designated in [1] to be non-hierarchical, we have implemented hierarchical coloured Petri nets with Renew by using synchronous channels. This means that a workflow can be refined hierarchically in our system, using the Petri net formalism.

We also distribute subnets across various processors via the construction of wrapper classes, called *stub classes*, which are used to set up the synchronous channel mechanism. The pair of channel invocations or synchronization requests that are required to start a subnet, and for control to be returned to its top-level net, are put in the body of a method within the subnet's wrapper class. Nets located on different machines can communicate via method calls [8, 10] and by means of CORBA (the Common Object Request Broker Architecture).

2.2. Process definition component

The process definition tool in our workflow system currently uses the Renew net editor, which is based on the JHotDraw package [4]. A net can be drawn in a drawing window and saved to a file in a serialized textual format. However there is no connection between separate drawings, *i.e.* there is no "point and click" mechanism for moving between levels in the hierarchy.

The Renew system assists in the construction of syntactically correct models by (a) making an immediate syntax check whenever an inscription is added or changed, (b) by providing menu options that when selected run extra checks (*e.g.* syntax checking and detecting naming conflicts), and (c) by not allowing the user to draw an arc between two places or two transitions.

2.3. Invoked workflow applications

When an individual Petri net transition is fired during the course of a model execution by the workflow engine, an application program may be accessed to carry out a

particular task. This may involve, for example, the saving of information to a database, the presentation and collection of information to and from a client's computer terminal, or the sending of some email messages. For those applications not already written in Java, we construct Java wrappers and access the wrappers by means of method calls (an *action* method) when a transition fires. Applications resident on remote computers are accessed by means of CORBA calls to the remote Java programs that provide the relevant services. Consequently application programs may be distributed across a network and accessed at the appropriate time by the workflow engine during workflow execution.

We illustrate how an application can be invoked by means of CORBA by a simple example (to increment a number on the server) described in Listings 1 and 2 and Figure 3.

Listing 1. Server class

```
package Incrementer;
public class IncrementImpl extends
  Incrementer._IncrementImplBase{
  private int sum;

  public IncrementImpl(java.lang.String name){
    super(name);
    sum = 0;
   System.out.println("Initial value of sum: "
      + sum);
  }
  public IncrementImpl(){
    super();
  }
  public int increment(){
    sum++;
    return sum;
  }
  public void sum(int val){
    sum = val;
  }
  public int sum(){
    return sum;
  }
  public static void main(String[] args){
    try{
      org.omg.CORBA.ORB orb =
        org.omg.CORBA.ORB.init();
      org.omg.CORBA.BOA boa = orb.BOA_init();
      Increment increment = new
        IncrementImpl("MyObject");
      boa.obj_is_ready(increment);
     System.out.println("Press return to exit
        server");
      try{
        System.in.read();
      }catch(Exception e){
        System.out.println(e);}
      boa.deactivate_obj(increment);
      orb.shutdown();
    }catch(Exception e){
      e.printStackTrace();}
  }
}
```

Figure 3. Increment net

Listing 2. Client class

```
package Incrementer;
public class IncrementClient{
  public IncrementClient(){
  }
  public static int incrementExecute(){
    String[] args = {""};
    org.omg.CORBA.ORB orb =
      org.omg.CORBA.ORB.init(args, null);
    Increment incrementer =
      IncrementHelper.bind(orb, "MyObject");
    incrementer.increment();
    return incrementer.sum();
  }
}
```

3. An example workflow

We consider the process of the enrolment of prospective postgraduate students at our university. This example has been chosen, because a number of university groups (i.e admission office, academic departments, financial office) must collaborate during this process, and the part of the process that each group is responsible for can be thought of and modeled as a sub-process. Each sub-process can be distributed to the "process-owner" group, while still enabling the whole process to be monitored by individual participants.

Figure 4 represents a Petri net model of the activities involved in the enrolment process. Each of these activities in turn can be further refined in separate sub-nets. For example, Figure 5 shows the refinement of one of these activities associated with the assessment of the students' proposed courses. In both of these figures, the net inscriptions have been omitted for clarity. Figures 4 and 5 demonstrate the usefulness of a hierarchical model in which the modeler can specify the basic sub-processes and their interactions without worrying about the details associated with each of the sub-processes.

A token in any instantiated net in our workflow model is a complex object representing an enrolment request from a prospective postgraduate student. This complex object has fields for identification and routing. Additional fields could be added to refer to documents and to record information (e.g. outcomes/decisions) associated with each request. The values of these fields are accessed or

set in the course of the workflow.

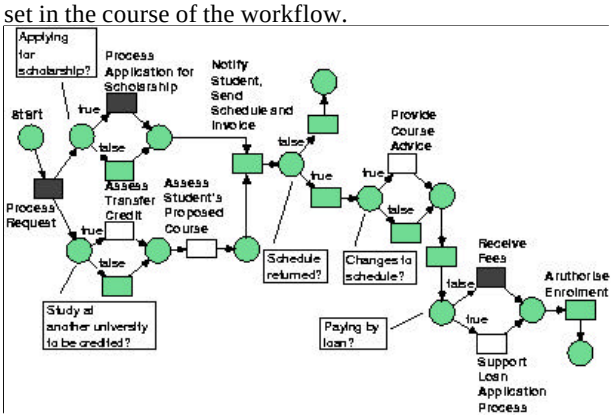

Figure 4. The enrolment net

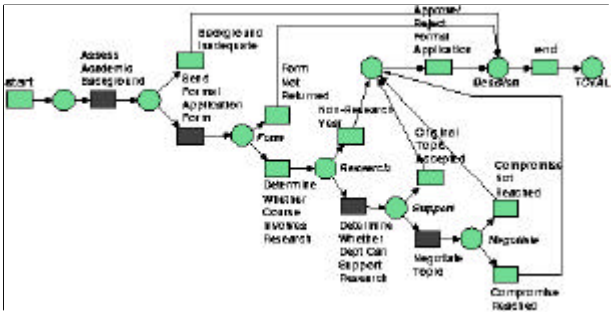

Figure 5. The admissions subnet

Figure 4 has three types of transitions: atomic tasks (black transitions), complex tasks (white) and routing and synchronizing tasks (grey). Every atomic task must have a resource (human or computer) allocated to execute it, whereas a complex task is a top-level abstraction for an underlying subnet, e.g. the complex task *Assess Student's Proposed Course* corresponds to the subnet in Figure 5.

In the enrolment net (Figure 4), an enrolment request is received to be processed. At this point two parallel activities can take place: (1) if the student has requested a scholarship, this application can be accessed and (2) if the student has course work that has been transferred from another university, this can be assessed. After the initial assessment of a student's record, the student's proposed courses to be undertaken must be evaluated. Figure 5 shows the corresponding activities associated with this evaluation. The outcome of this evaluation, which specifies a set of papers or a research plan to be undertaken along with the result associated with the transition *Process Application for Scholarship,* is sent to the student. The student can then either accept the proposed program of study or decline the offer. If the student would like to make some changes to the proposed program of study, approval must be obtained, which is represented in the model by the transition called *Provide Course Advice*. Finally the student associated with the request is officially enrolled once fees have been received or have been arranged to be paid by loan.

As mentioned in section 2.1, the relationship between two Petri nets in a hierarchical model is such that a transition in the top-level net can be substituted for by a subnet – here, the transition and the subnet that replaces it share the same input place(s) and output place(s). The actual implementation of this concept in Renew nets uses synchronous parameterized channels. When the transition *Assess Student's Proposed Course* (ASPC) fires, it synchronizes with the *start* transition in Figure 5 and the two transitions effectively share the invoking transition's input place. The ASPC transition is blocked until the subnet execution is completed and the returned request is deposited in the ASPC transition's output place via the synchronization between the invoking transition and the subnet's *end* transition.

In the sub-net diagram shown in Figure 5, first an initial assessment of the student's academic background is made, based on the student's course proposal and evidence of an academic qualification. The student can then lodge a formal application indicating one or two areas of interest (*Send Formal Application Form*). The application is then evaluated to check whether it involves course work or solely research. If the program of study involves research, then a decision must be made as to whether a faculty member can support (supervise) the research in the proposed area of interest (*Determine Whether the Dept. Can Support Research*). The research may have to be modified in order to locate a staff member who is willing to supervise the student undertaking the specific area of research (Negotiate Topic).

Finally, the proposed program of study (either course work or research) must be approved (or rejected) by the postgraduate committee (*Approve/Reject Formal Application*). The output of this transition indicates the decision that has been made with regard to a particular application and completes the task associated with this subnet. In this paper we are explicitly describing the refinement detail of only one of the subnets associated with the top-level net. Refinement of the other subnets is done in a similar fashion.

The next phase in the prototype development was to install CORBA communication in the Java subnet classes so that the Renew net-subnet communication could be conducted in a distributed environment. In this circumstance getting references to the subnet objects was no longer simple, because CORBA server-client interaction was required and each CORBA server had to do the following in order to export a subnet object that was simulatable:

(a) use a Renew ShadowSimulator instance to load the shadow form of the net. (In Renew a net is simulatable in two forms, graphical or shadow. The graphical form is desirable in our resource project because one of our goals is to make it possible for a process-owner to

monitor his or her subnet and eventually be able to alter its structure. However use of the graphical form necessitates manual loading by opening it in the net editor which does not work in a distributed environment.)

(b) use a Renew ConcurrentSimulator instance to put the shadow net into simulation mode.

The token class, of course, also had to be CORBA-enabled.

Having successfully maintained the communication mechanism in a distributed environment we were confronted with the problem of not being able to reconcile the shadow form of the subnets with their graphical representations – required for monitoring the rate of throughput of multiple tokens and evaluating "what-if" scenarios. As an intermediary solution to this problem, we are using a combination of bar charts and the Renew Event package to monitor the throughput (numbers of tokens in key named places per unit time) in each subnet. For example, Figures 6 and 7 show charts associated with two of the subnets in our workflow model – they were screen-dumped during a simulation. The *TOTAL* place was included in order to determine how many requests in total passed through each subnet.

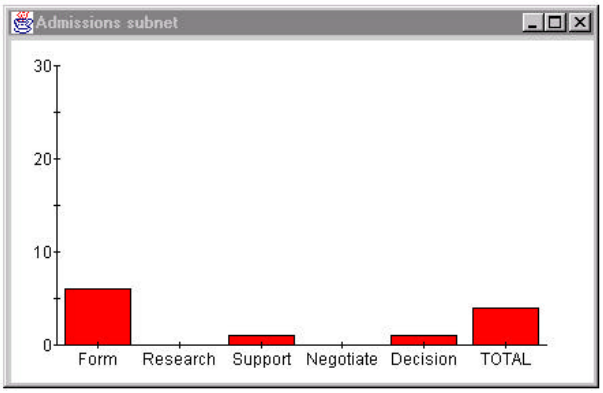

Figure 6. Dynamic chart of the Admissions subnet

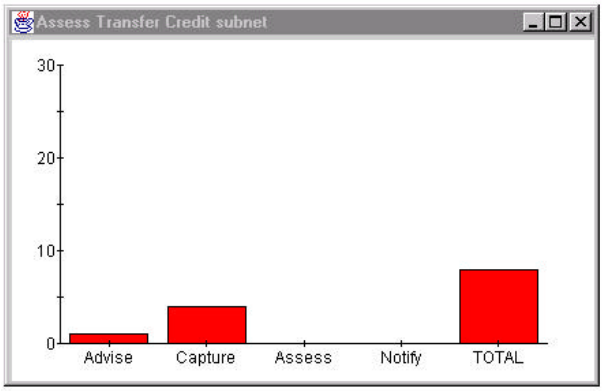

Figure 7. Dynamic chart of the Assess Transfer Credit subnet

4. Towards adaptive workflow

In many situations processes, resources, and the constraints associated with various businesses and organizations that we are trying to model are changing frequently. The architecture of workflow systems should be sufficiently flexible to cope with these unpredictable changes. Since a change in one component of the system can have some impact on the rest of the process, these changes should be explicitly represented on the overall process model which could be viewed by all the participants of the system. In a distributed workflow system, where each section of an organization might be in a different geographical location, designated regional representatives should be able to modify some aspects of their sub-process if they need to do so. At the same time the interaction between various sub-processes should be managed by maintaining an overall organizational model that provides dynamic links to distributed, changeable sub-processes. This resulting overall model can reveal any inconsistencies or any other problems which can arise due to resource conflicts.

Changes to the workflow can be either static or dynamic. Static changes refer to those changes made to the workflow while it is not being executed. Modification can be made to various elements of workflow, such as the process, the available resources, as well as the changes that can be made to the resource allocation mechanism. Dynamic changes refer to those changes made to the active instances of the workflow [12].

In the workflow literature, various categories of adaptability have been defined [12] such as *flush*, *abort*, *migrate*, *adapt*, and *build*, with each of these terms representing a respective increase in the extent to which the system adapts to change. In flush mode situations all current instances are allowed to complete according to the old process model, but new instances are planned to follow the new model. For the other four modes, the existing, active instances of the workflow can be impacted by the change. In the *abort* mode the current active instances are aborted, in the *migrate* mode, the execution of the workflow continues while the new changes are integrated into the process, in the *adapt* mode the process must be altered for individual instances in order to accommodate some exceptional cases, and in the *build* mode the whole process can be rebuilt at runtime so that the appropriate process model that corresponds to the particular situation at hand can be created.

Our approach is to use the *migrate* mode. Each workflow is an instance of a workflow template (a Java class). When a new workflow (template) is produced, old instances are allowed to execute to completion if their tokens occupy places that have been changed under the new arrangement. Otherwise a new instance of a model is produced and the tokens are inserted into the appropriate

places. The system keeps track of multiple models to accommodate both old and new workflow instances and their job tokens. In such a scenario the old model is linked with the new model by means of sharing the same resources. As the old job tokens are completed old model instances can be discarded.

An example of how the *migrate* mode works is shown in Figure 6.

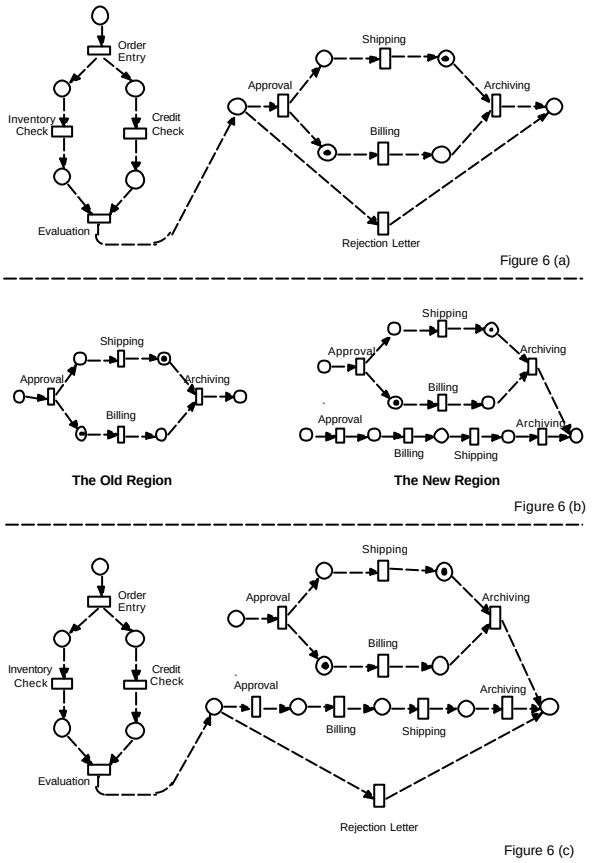

Figure 6. An adaptive Petri net example

Figure 6 (a) shows an existing workflow associated with the receipt of an order and its processing (adapted from [3]). Shipping of merchandise and billing are carried out in parallel. It is subsequently decided to carry out billing and shipping sequentially. At the time that this change is made to the model, we may have some existing tokens that are already in the parallel segment of the old model. In order to accommodate this change dynamically, we must include the existing tokens in the old part of the model as a part of the new, combined model until their tokens complete their transit through the workflow (as in the case of *flush* mode), as shown in Figure 6 (c). This is achieved by making sure that separate workflow instances are coupled by means of synchronous channels.

However if there are existing tokens only in the "early" part of the model that is unaltered in the new model, then we just "migrate" to the new model. This migration is accomplished by saving the state information of the existing instance of the workflow and importing that information into a new instance of the new model which has the sequential arrangement of the *Billing* and *Shipping* tasks.

5. Conclusion

This paper describes a framework for a distributed adaptive workflow system. A prototype of the system has been developed where the process definition can be represented by means of a Petri net formalism. The workflow engine employs coloured Petri nets and uses the Renew implementation as a system component that links with workflow applications by means of CORBA technology, enabling access to remote clients and servers.

We are planning to extend the current prototype with the following capabilities:

- Provide support for the *Workflow Client Application* and *Invoked Applications* components of the WfMC Reference Model (Figure 1)
- Use CORBA services such as the Naming, Trading, Event, and Persistent Services in order to maintain a flexible system architecture
- Provide timed tokens in order to measure the performance of the system and examine the extent to which the process deadlines have been met
- Provide utilities for graphically monitoring system status and performance throughput.

Further research activities involve:

- Exploring the possibility of using third party tools for analyzing the properties of the model such as reachability, boundedness, deadlocks, and liveness
- Using agent technology in order to adapt the process in response to various agents which can be made responsible for monitoring the system from different points of view, such as resource allocation.

We emphasise that the work described here will apply not just to WFMSs as they are currently applied in the business community, but to the larger scheme of adaptive, distributed software process execution, where growing software interoperability means that complex tasks may be executed across a distributed environment. It is for this reason that distributed agent technology is likely to play an increasingly important role in the development of workflow architectural frameworks such as described in this paper.

Acknowledgements

The work reported in this paper has been funded by an Otago Research Grant entitled "Adaptive Workflow Modelling in a Distributed Environment". We wish to acknowledge the practical and theoretical support we have received from Olaf Kummer, an authority on Renew, and the contribution of Dr Stephen Cranefield as a project advisor.

References

[1] R. Bastide, D. Buchs, M. Buffo, F. Kordon and O. Sy, *Questionnaire for a taxonomy of Petri net dialects* <http://www-src.lip6.fr/homepages/Fabrice.Kordon/PN_STD_WWW/Qresult.html>

[2] S. Christensen and N. Damgaard Hansen, "Coloured Petri nets extended with channels for synchronous communication", *LNCS 815, Application and Theory of Petri Nets 1994, Proc of 15th International Conference, Zaragoza, Spain, June 1994*, R. Valette (ed.), pp. 159-178, Springer-Verlag, Berlin, 1994

[3] C.A. Ellis, K. Keddara and G. Rozenberg, "Dynamic change within workflow systems", *Proc of Conference on Organizational Computing Systems (COOCS'95), Milpitas, CA, August 1995*, pp. 10-21, ACM Press, New York, 1995

[4] E. Gamma, *JHotDraw*, 1998 <http://members.pingnet.ch/gamma/JHD-5.1.zip>

[5] M. Hammer and J. Champy, *Reengineering the Corporation*, Harper Business, New York, 1993

[6] Y. Han, A. Sheth and C. Bussler, "A taxonomy of adaptive workflow management", *1998 ACM Conference on Computer Supported Cooperative Work (CSCW-98), Seattle, WA, November 1998* <http://ccs.mit.edu/klein/cscw98/paper03>

[7] K. Jensen, *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use, Vol. 1: Basic Concepts*, Springer-Verlag, Berlin, 1992

[8] O. Kummer and F. Wienberg, *Renew – User Guide, Release 1.*3 University of Hamburg, Department for Informatics, Theoretical Foundations Group and Distributed Systems Group, September 2000 <http://www.informatik.uni-hamburg.de/TGI/renew/>

[9] O. Kummer, "Simulating synchronous channels and net instances", *5. Workshop Algorithmen und Werkzeuge für Petrinetze*, J. Desel, P. Kemper, E. Kindler and A. Oberweis (eds.), pp. 73-78, Forschungsbericht Nr. 694, Universität Dortmund, Fachbereich Informatik, October 1998

[10] O. Kummer. "Tight integration of Java and Petri nets", *6. Workshop Algorithmen und Werkzeuge für Petrinetze*, J. Desel and A. Oberweis (eds.), pp. 30-35, J.W. Goethe-Universität, Institut für Wirtschaft-informatik, Frankfurt am Maim, October 1999

[11] P.D. O'Brien and W.E. Wiegand, "Agent based process management: applying intelligent agents to workflow", *The Knowledge Engineering Review*, 13(2):1-14, June 1998

[12] S.W. Sadiq, O. Marjanovic and M.E. Orlowska, "Managing change and time in dynamic workflow processes", *International Journal of Cooperative Information Systems*, 9(1-2):93-116, World Scientific Publishing Company, 2000-09-13

[13] W.M.P. van der Aalst, "The application of Petri nets to workflow management", *The Journal of Circuits, Systems and Computer*, 8(1):21-66, 1998

[14] W.M.P. van der Aalst, "Three good reasons for using a Petri-net-based workflow management system", *Proc of International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, S. Navathe and T. Wakayama (eds.), pp. 179-201, Camebridge, Massachusetts, November 1996

[15] Workflow Management Coalitition, *The Workflow Reference Model*, Document No. TC00-1003, Issue 1.1, 1995