An Adaptive Distributed Workflow System Framework

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 are increasingly used to assist the automation of business processes that involve the exchange of documents, information, or task execution results. Recent developments in distributed information system technology now make it possible to extend the workflow management system idea to much wider spheres of activity in the industrial and commercial world. 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. Key technical elements of the framework include the use of coloured Petri nets and distributed object technology (CORBA).

1. Introduction

In recent years developments in networking and telecommunications have opened up enormous opportunities for linking up disparate information sources and computational modules. This has led, on the one hand, to the development of distributed information systems that integrate dispersed information sources. On the other hand, considerable interest has been generated in the area of software interoperability: the linking of software modules to carry out complex computational tasks. An example of this second type of software integration is that of Workflow Management Systems, which provide 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 workflow management software systems, 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. (One example of such a business process is the software engineering process, itself, and we will discuss this example further on in this paper.) 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 workflow management systems 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 software engineering,
- plans for adaptive workflow modelling, and
- conclusions and future work.

2. System architecture

The Workflow Management Coalition's Workflow Reference Model (Figure 1.) shows the major components and interfaces within the architecture of a workflow management system (WFMS) [17], 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 Workflow Management Coalition 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 dataflow mechanisms and distributed systems [15, 16]. In addition they have a graphical notation that is both intuitive and rigorous. Because of their formal properties, there are a number of techniques available for the quantitative analysis of their behaviour.

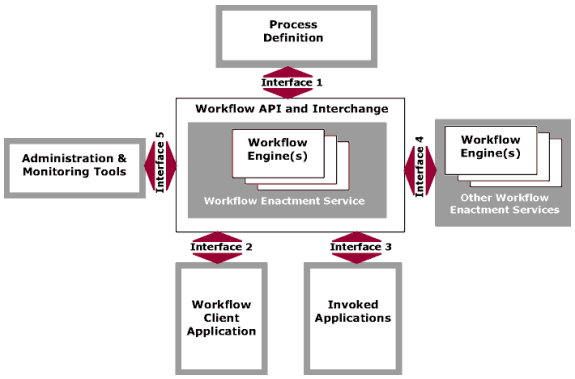

Figure 1. Reference architecture for the Workflow Management Coalition.

Petri nets consist of a net structure and a set of rules defining the behaviour of the net. We employ coloured Petri nets, which represent a powerful, high-level generalisation of conventional Petri nets [7]. The net structure is a bipartite, directed graph, $S = (\mathsf{E},P,T,A,E,G)$, where

- E is a finite set of types (colours)
- P is a finite set of places,
- T is a finite set of transitions,
- A is the set of arcs (),
- E is a set of arc expressions that map each arc and variable value from the scope of its associated transition into a multiset of typed (from E) values,
- G is a set of guard functions that map variables in the scope of a transition into a type boolean.

Petri nets have an initial marking, M_0, of the set of places that indicate typed (coloured) tokens located at those places. For each transition, arcs directed into that transition (its "input arcs") identify places ("input places") whose marking contain token values that are within the variable scope of that transition. The behaviour of a net is governed by rules that determine when a given transition can fire. When a transition fires, a marking of the net, M, is transformed into a new marking, M', that indicates a new distribution of tokens over the Petri net places. In general for a transition to fire, its guard must evaluate to TRUE and its input arc expressions must evaluate to non-empty colour values. More details concerning the behaviour of coloured Petri nets can be found in [7].

Significant reasons for preferring Petri net modelling in connection with process and workflow modelling over other notations are [16]

- 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 Workflow Management Coalition [16].
- 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 [16] make it possible to identify 'dangling' tasks, deadlocks, and safety issues.

2.1 Workflow engine component

We use the Renew (Reference Net Workshop) [14] modelling tool to represent and execute the workflow model. Renew employs a variant of coloured Petri nets (which are called "reference nets") and supports "synchronous channels", which will be described below. 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. In Renew, Petri net arc inscriptions are Java expressions and token types (colours) can be any Java class. Transition expressions are also available in Renew, which are Java expressions that are evaluated at the time of transition firing. A transition expression, for example, can use the equality operator, `=`, to affect the binding of variables within the variable scope of the transition. An example Renew Petri net is shown in Figure 2. Petri net places are shown as circles, and transitions are shown as rectangles.

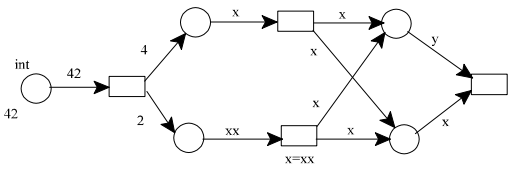

Figure 2. A coloured Petri net.

In Figure 2 [8] a token in the leftmost place has a value of 42. The leftmost transition takes this token as an input and deposits a token with a value of 4 and another token with a value of 2 in two separate places. The succeeding transitions bind the variables `x` and `xx` to the values of the incoming tokens. Each transition has its own variable scope, so that `x` is bound to different values for the two middle transitions.

When a Petri 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" [3,8,9], 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 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 subordinate net 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 subnetwork. Although the reference nets of Renew are designated in [2] 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 access 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 and by means of CORBA (the Common Object Request Broker Architecture) [8, 10].

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 [8]. A net can be drawn in a drawing window and saved to a file in a 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();}

  }//main

}//IncrementImpl
```

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");
    for(int i=0;i<1000;i++){
      incrementer.increment();
    };
    return incrementer.sum();
  }

}//IncrementClient
```

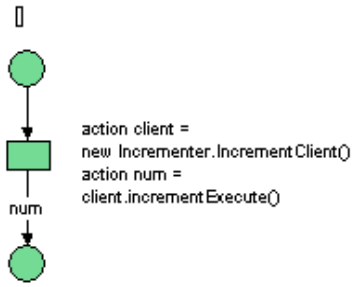

Figure 3. Increment net.

3. An example workflow

We consider an example application in the area of software engineering itself – the software development process and workflows associated with it. Several tools have been developed specifically to support the process modelling of software engineering [13]. Software engineering is an interesting workflow problem area, because software development is a difficult, human-intensive activity that often involves distributed, concurrent activities, and processes are often subject to changing circumstances, such as changing requirements or available technology [13].

3.1 The software change process

For our example, we consider one component of software development: the software change process, which we have adapted from [1]. The underlying process involves the modification of a program unit (for example, a Delphi unit which is divided into interface and implementation

sections) caused by changes in the global specification of a system. Our top-level net shown in Figure 4 has a single task called `Change unit`, which serves as the driver for the subnet shown in Figure 5. It is in the subnet that the actual change process takes place. Additional tasks from the maintenance and enhancements phase of the overall system development life cycle could be added to the top-level net, and each task implemented by an additional subnet.

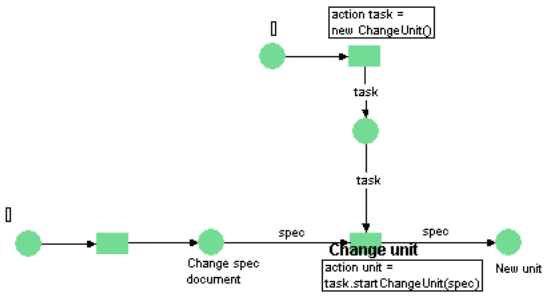

Figure 4. The software change process Petri net (top-level view).

Assume that a document specifying a required change to be made to the global system specification has been received, and the specific unit that is affected by this change has been identified (Figure 4). The refinement of the `Change Unit` process is shown in Figure 5. This is an example of hierarchical Petri net refinement and presents six basic tasks: `Design the change`, `Propose interface change`, `Unit coding`, `Generate test cases`, `Compile` and `Run tests`.

The communication between the two nets shown in Figures 4 and 5 is conducted via synchronous channels. A wrapper class was created for the subnet of Figure 5, using the Renew StubGenerator and StubCompiler. The interface to the wrapper class contained a single triggering method specifying the type of the arguments to be passed in (a document) and out (a unit). Two synchronization requests (downlink inscriptions) were placed in the method body to synchronize the invoking transition (`Change Unit`) in the top-level net (Figure 4) with the initial transition in the subnet (Figure 5) via the channel, `startChangeUnit`, and with the final transition in the subnet via the channel `result`. The second channel is needed to make the `Change unit` transition wait until the method call completes and produces a token to deposit in the `New unit` output place. An explicit method call identifier (`instance`) is recommended (but not required) in both synchronization requests to handle concurrent method calls.

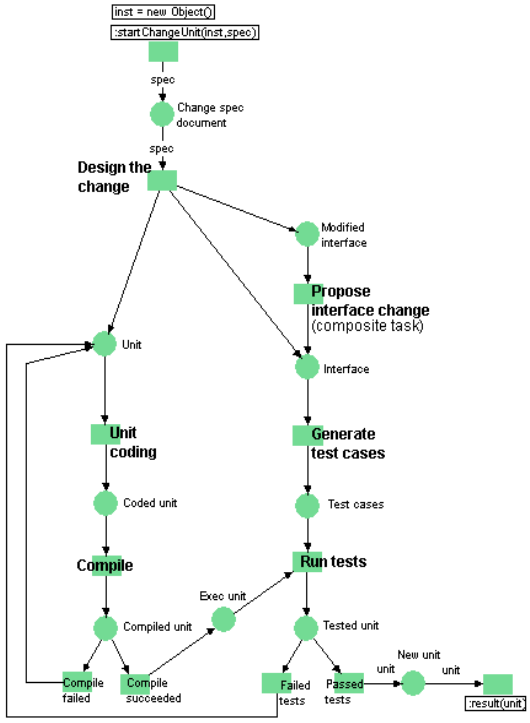

Figure 5. The unit change process refinement subnet accessed from net of Figure 4.

It is assumed that during the `Design the Change` task, there will always be a change to the software unit implementation and that it will involve a change to the code. A copy of the interface must be extracted so that test cases can be generated from. The distinction between whether the `Design the Change` task modifies the interface of the original unit or not, is made because a modified interface must be routed to the `Propose Interface Change` task, a composite task whose implementation is not shown here (it concerns the requirement that all designers who use a given unit agree on any new interface definition for the unit before a change operation can proceed).

If the re-coded unit is compiled successfully, the test cases are executed in connection with that compiled unit. Otherwise, the unit is returned to the `Unit Coding` task to be re-coded.

The change process is terminated when the unit passes the test.

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 has 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.

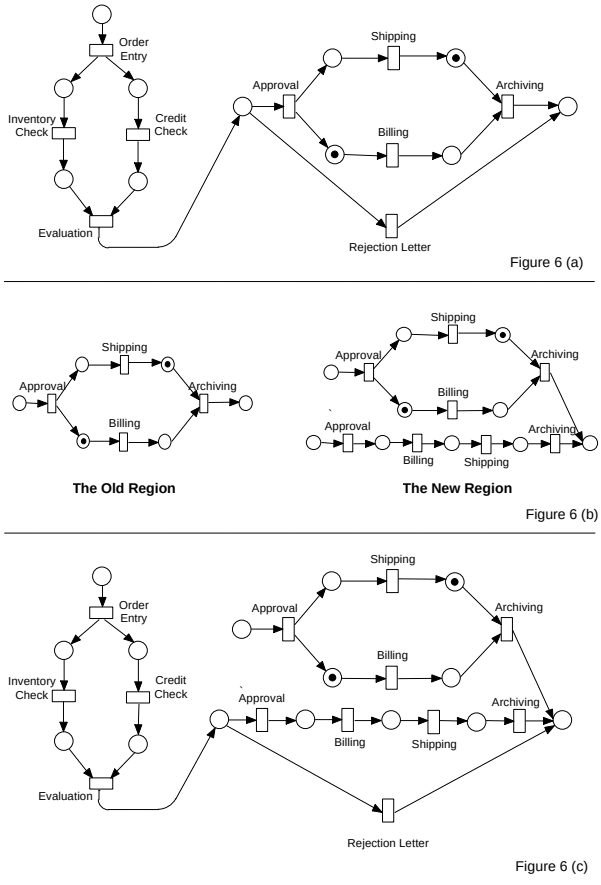

Figure 6. **An adaptive Petri net example.**

Figure 6 (a) shows an existing workflow associated with the receipt of an order and its processing (adapted from [4]). 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 system with the following capabilities:

- 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 workflow management systems 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] S. Bandinelli, A. Fuggetta, C. Ghezzi and L. Lavazza. "SPADE: An environment for software process analysis, design, and enactment". A. Finkelstein, J. Kramer and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pp. 1-8. Research Studies Press, 1994.

[2] 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/questionnaire.html (Results of the questionnaire: http://www-src.lip6.fr/homepages/Fabrice.Kordon/PN_STD_WWW/examples.html)

[3] S. Christensen and N. Damgaard Hansen, "Coloured Petri nets extended with channels for synchronous Communication". Valette R., editor, *Application and Theory of Petri Nets 1994, Proceedings of the 15th International Conference, Zaragoza, Spain*, volume 815 of *Lecture Notes in Computer Science*, pp. 159-178. Springer-Verlag, Berlin, 1994.

[4] C.A. Ellis, K. Keddara and G Rozenberg, "Dynamic change within workflow systems". N. Comstock, C. Ellis, R. Kling, J. Mylopoulos and S. Kaplan, editors, *Proceedings of the Conference on Organizational Computing Systems (COOCS'95), Milpitas*, pp. 10-21, ACM Press, New York, 1995.

[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 Conference on Computer-Supported Cooperative Work (CSCW-98), Seattle, WA*, 1998. http://ccs.mit.edu/klein/cscw98/

[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.*

http://www.informatik.uni-hamburg.de/TGI/renew/

[9] O. Kummer, "Simulating synchronous channels and net instances". In J. Desel, P. Kemper, E. Kindler and A. Oberweis, editors, *5. Workshop Algorithmen und Werkzeuge für Petrinetze*, pp. 73-78. Forschungsbericht Nr. 694, Universität Dortmund, Fachbereich Informatik, October 1998. http://www.informatik.uni-hamburg.de/TGI/publikationen/biblio_kummer_eng.html

[10] O. Kummer. Tight Integration of Java and Petri nets. In Desel, J. and Oberweis, A. editors, *6. Workshop Algorithmen und Werkzeuge für Petrinetze*, pp. 30-35. J.W. Goethe-Universität, Institut für Wirtschaft-informatik, Frankfurt am Main, October 1999. http://www.informatik.uni-hamburg.de/TGI/publikationen/biblio_kummer_eng.html

[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.

[13] R.A. Snowdon and B.C. Warboys, "An introduction to process-centred environments". A. Finkelstein, J. Kramer and B. Nuseibeh, editors, *Software Process Modelling and Technology*, pp. 1-8. Research Studies Press, 1994.

[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.
http://www.informatik.uni-hamburg.de/TGI/renew/

[15] W.M.P. van der Aalst, "The application of Petri nets to workflow management", *The Journal of Circuits, Systems and Computers*, 8(1):21-66, 1998.
http://wwwis.win.tue.nl/~wsinwa/publications.html

[16] W.M.P. van der Aalst, "Three good reasons for using a Petri-net-based workflow management system". S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pp. 179-201, Cambridge, Massachusetts, Nov 1996.
http://wwwis.win.tue.nl/~wsinwa/publications.html

[17] Workflow Management Coalition, *The Workflow Reference Model*, Document Number TC00-1003, Issue 1.1 (1995), p. 20.
http://www.aiim.org/wfmc/mainframe.htm