An Agent–Based Workflow Management System

Krzysztof Palacz and Dan C. Marinescu
Computer Sciences Department
Purdue University,
West Lafayette, IN, 47907
email: {palacz,dcm}@cs.purdue.edu

Abstract

In this paper we discuss the architecture of an agent–based workflow management system built around the Bond agent framework. We address the problem of mapping a workflow description into the Blueprint language used for agent description. Bond agents can be modified dynamically by changing the data structure controlling the scheduling of actions in the multi-plane state machine model of the agent. The modified blueprints can be generated after the surgery of an agent. From the modified blueprint we can create the modified workflow description and complete the cycle supporting dynamic workflows.

Contents

Introduction

Workflow systems are designed to automate complex activities consisting of many dependent tasks. Workflow management systems, WFMS, are widely used to automate business processes (Alonzo *et al.* 1997), and there is growing interest in their application to data-intensive scientific and engineering problems (Wainer *et al.* 1996). There is ample evidence that the business use of workflows increases productivity and improves the quality of the products of an organization and we should expect similar benefits for scientific and engineering applications, e. g. for tasks that involve data collected from many sites and intricate computations to extract useful information from the data.

Business-oriented workflow management systems, can be used for (Alonzo *et al.* 1997): (a) administrative tasks involving well established sequences of steps e.g. publishing books in a publishing house, (b) ad-hoc tasks with frequent occurrences of exceptions e. g. preparing books for publication by an individual author (various institutions may have similar yet different requirements and guidelines), (c) collaborative tasks requiring frequent interactions among participants and many iterations over the same step or even repetitions of previously accomplished steps, e. g. writing a book by several co-authors. Typically, WFMS support processing of documents, of mail, or are concerned with coordination of legacy and newly developed applications. There exists a variety of commercial offerings, usually not developed from ground up as WFMS but built on top of applications designed for other uses, such as image-processing, e. g. FileNet's WorkFlo, relational database management systems, e. g. IBM's FlowMark and computer supported collaborative work e. g. Lotus Notes. The interest in WFMS led to the creation of the Workflow Management Coalition (Workflow Managment Coalition 1996) that brings together designers and users of WFMS systems.

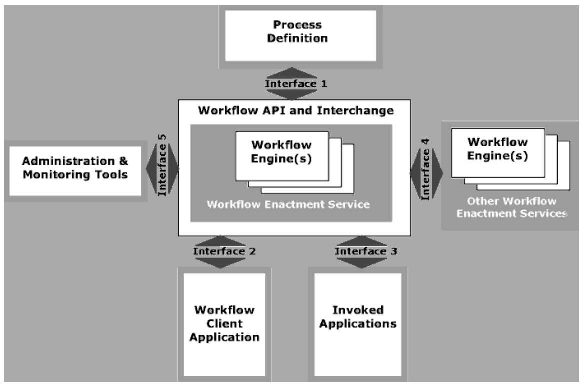
Process Definition
Interface 1
Workflow API and Interchange
Administration & Monitoring Tools
Interface 5
Workflow Engine(s)
Workflow Enactment Service
Interface 4
Workflow Engine(s)
Other Workflow Enactment Services
Interface 2
Interface 3
Workflow Client Application
Invoked Applications

Figure 1: Reference model of a Workflow System introduced by the Workflow Management Coalition (Workflow Managment Coalition 1996)

The basic architecture of a workflow system is depicted in Figure 1 and consists of several components: workflow/process definition module, the workflow engine(s), support and monitoring tools, various client applications. Most existing systems are RDBMS-based and, in theory, can take advantage of the reliability, availability and scalability features of the underlying database system. However, in practice workflow systems require advanced transaction models not supported by most commercial DBMS. Consequently, an Workflow Management System built on top of the DBMS does not exhibit desired properties (Alonzo *et al.* 1997). Moreover, many of the tasks that a workflow system might be used to automate are non-transactional in nature (e. g. interaction with humans, integration of applications that do not provide transactional semantics support), which further limits the benefits of basing a workflow management system on a DBMS (Worah & Sheth 1996). It is also worth noting that the typical usage profile of a WFMS is dissimilar to those of a RDBMS. RDBMS systems usually are subject to many read requests and relatively few updates while in the case of WFMS the queries are not very elaborate (do not require sophisticated data query language features) and updates occur with higher frequency.

Uses of agents in WFMS have been discussed in several publications e. g. in (Chang & Scott 1996; O'Brien & Wiegand 1998). Usually in WFMS implementations agents act as personal assistants performing actions on behalf of the workflow participants and/or facilitating interaction with other participants or existing WFMS. In this paper we propose an agent–based architecture for workflow enactment and for the monitoring components of a workflow management system. In particular we concentrate on the use of agents as *case managers*: autonomous entities overlooking the processing of single units of work. Our assumption is that an agent–based implementation is more capable of dealing with dynamic workflows and with complex processing requirements where many parameters influence the routing decisions and require some inferential capabilities. We also believe that the software engineering of workflow management systems is critical and instead of creating monolithic systems we should assemble them out of components and attempt to reuse the components.

Recently, we proposed a multi – plane state machine agent model (Bölöni & Marinescu 1999a) and released a component – based architecture for building agents. Bond agents were used to construct a network of PDE solvers (Tsompanopoulou *et al.* 1999), to support adaptive resource allocation for multimedia applications (Jun *et al.* 1999), and for several other projects.

This paper is organized as follows. In Section we briefly describe the multi – plane state machine agent model we have proposed (Bölöni & Marinescu 1999a) and implemented in the Bond system (Bölöni & Marinescu 1999b). Then, we review briefly various workflow models in Section and the Workflow Definition Language in Section . The workflow management architecture we propose is presented in Section .

A Multi-Plane State Machine Agent Model

In this section we present a model for software agents. An agent is a composite object consisting of: (1) a set of planes, P, representing n concurrent activities, and (2) a model of the world, W containing information about the environment.

$A = \{P, W\}$
$P = \{P_1, P_2, ...P_n\}$

Each concurrent activity is described by a state machine, P_j, it consists of a set of states S_j with $|S_j|$ elements, S_i^j , a set of transitions, T_i with $|T_i|$ elements T_i^j, and a set of strategies, θ_j with $|\theta_j|$ elements, θ_i^j. Each state has one strategy associated with it, thus $|\theta_i| = |S_i|$.

$P_j = \{S_j, T_j, \theta_j\}$

We say that P_j is in state S_i^j if the strategy associated with this state, θ_i^j has completed and reported either success or failure. Each strategy θ_i^j, reveals an *interface* κ_i^j consisting of a subset of its state variables. The actions carried out by strategies are *atomic*, either all state variables in the agent's interface are updated or none is. Strategies have a *bounded execution time*, a strategy θ_i^j either terminates within τ_i^j units of time or aborts.

The model of the world is a passive object consisting of the intersection of the κ_i^j interfaces and the agent state vector reflecting the state of each individual state machine, $\phi = (\phi_1, \phi_2, ..\phi_n)$.

$W = (\kappa, \phi)$

All actions carried out by an agent are the result of the execution of the strategies in the set $\{\theta_i^j\}$. Thus an agent composes the $\{\theta_i^j\}$ strategies into coherent actions such that n of them are being carried out concurrently.

The *blueprint* of an agent is a textual description of all the components presented above. Agents are created from blueprints by dedicated objects called *agent factories*.

Single plane agents support multiple threads of control confined to a single strategy, they do not support independent activities running concurrently. Designing reactive agents able to respond to external events could be cumbersome and inefficient when we are restricted to single plane agents. External events must be queued and the reaction time, can be substantial because the agent must reach a state when it is capable to respond to external events.

Yet single plane agents can be very useful and perform complex functions. For example consider the following specification of a data acquisition and analysis, DAA process: a set of m sensors provide raw data that needs to be checked for consistency. If the consistency test succeeds, the raw data is archived and then a modeling program is activated, else a new set of raw data is collected. The consistency check, requires an inference engine, it is based upon a set of rules that guarantee that at least m_1, $m_1 < m$ sensors function correctly, that precisely $m_2, m_2 < m_1$ critical sensors are among those functioning correctly, and that the data provided by the correct sensors have statistical characteristics within a given range. Once the results of the consistency test are known, a procedure is activated to select the site(s) where the computer models will run. This selection procedure is also rule-based, up to date status information about the systems described in a resource file is collected. Depending upon the number of data sets in the batch, we select a subset of systems in the resource file as modeling site(s). Finally, a data staging phase replicates the raw data sets to the modeling site(s) and the modeling program is activated at each of the selected sites.

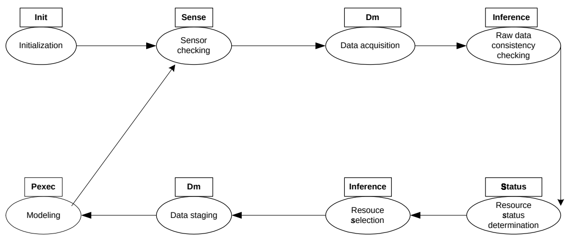

Figure 2: Data Acquisition and Analysis Agent

Figure 2 shows the state transition diagram of the *DAA* agent. For sake of simplicity we do not show the error recovery states. The agent starts in an initialization state. The *Init* strategy loads into the agent's model the sensor configuration file, the resource file, the rules and facts for the consistency check, the rules and facts for selecting the modeling sites, and possibly other information. In the sensor state, the agent executes the *Sense* strategy and visits all the sites specified in the sensor configuration file to determine if the raw data sets are available. Then the system moves into the data acquisition state, it executes the *data migration, Dm* strategy, and collects all raw data sets. In the next state we perform the consistency test. We run the *Inference* strategy with the proper sets of facts and rules available from the agent's model and if successful the strategy writes into the agent's model the information necessary to identify the selected sensors and the raw data sets they provided. In the next state we evaluate the status of the resources in the resource file using the *Status* strategy. Then we select the target systems for modeling using again the *Inference* strategy but with a different set of rules and facts. Next we move into a data staging phase and execute the *data migration, Dm* strategy with a different set of sources, destinations, and data sets. Finally we enter the modeling state where a *program execution, Pexec* strategy is used to trigger the execution of the modeling program.

This example illustrates important aspects of the methodology for agent design presented in (Bölöni & Marinescu 1999a). The *Data migration* and the *Inference* strategies are reused. Every time a strategy is instantiated it updates its internal state with data from the model and upon completion updates the model.The agent's model serves as a shared memory for all strategies of a state machine. In the case of a single plane agent the access to the model is strictly sequential, no race conditions may occur. Individual strategies exporting variables in its interface may associate a capability with each variable and limit access to it only to strategies granted the capability.

The methodology encourages a hierarchical design. Some of the strategies presented above e.g *Sense* and the *Status* are themselves composed of strategies that may run concurrently to explore the status of each sensor or report performance data from individual systems.

The methodology encourages a gradual approach to complex system design (Bölöni & Marinescu 1999c), (Bölöni *et al.* 1999). In our example we design and test first a *DAA* agent capable to run the modeling program on a fixed target system. Then we add the dynamic selection strategy for a single target system. Finally we replace the *program execution, Pexec* strategy with one allowing cooperation among a group of modeling programs.

The visitor pattern can be used to accommodate cases when code mobility is undesirable. For example proprietary code may be used to report the resource status. Thus the *resource status, Status* strategy may require the agent to visit individual sites and at each site activate a strategy available only locally. Last but not least this example shows that agent autonomy and mobility are complementary dimensions of agent design.

The behavior of a Bond agent can be altered dramatically while the agent is running by adding new planes or by altering the state machine of an existing plane. This process, called agent surgery, modifies the data structure controlling the scheduling of various activities of the state machines of the agent's planes. Once the surgery has been performed the modified blueprint can be generated. For example, in case of the data acquisition and analysis agent, instead of using a visitor pattern for checking the sensors, we can modify the agent by adding one plane for monitoring each sensor and for performing some data reduction locally. In this case the planes of the agent would run ad different sites.

This dynamic modification of the agent may be triggered by timing consideration, e. g. when the number of sensors exceeds a certain threshold it may be impractical to check them one after another and create planes that check the status of the sensors concurrently.

In this framework, we can construct the new workflow specification from the blueprint of the modified agent.

Workflow models

How to build the case managers using the Bond framework? We are looking for a conceptual framework to

- design/define/analyze workflows, and
- realize workflows at runtime.

Though we are primarily interested in the second objective, it seems advantageous to use the same framework for both steps, since we cannot ignore the design/analysis issue and let some external tools handle it because ultimately we are interested in allowing "sensible" dynamic changes of the workflow. We wish to express parallel, sequential, and conditional execution as well as any combinations thereof. In any case a workflow can be presented as a directed graph with certain nodes marked in some way to denote the state of execution at a given time. Several approaches are possible:

- Activity charts – nodes are activities, arcs can be labeled by the data transferred.
- State charts – nodes are states, arcs are labeled by the activities (actions) to be performed.
- Petri nets – both activities and states are represented by nodes (Petri net transitions and places, respectively)

The activity charts based model requires that the routing information be associated in some way with the action. An example here is the model presented in the Workflow Definition Language specification, described in Section . This approach does not seem to be very appropriate for our purposes because if we were to map activities into Bond strategies then we would have to combine routing functionality (specific to a particular workflow) and domain functionality (very often generic in nature) in one strategy thus considerably limit reusability of strategies. This approach is also not very well suited to model activity triggers and milestones (see (Van der Aalst 1998)).

An example of the state – based approach is reflected by the statechart concept (Harel *et al.* 1987), where actions may be initiated and stopped whenever the system performs a transition from one state to another. More specifically, each transition has associated with it the triple (Event, Condition, Action). The action is performed when the transition takes place, namely when the specified event occurs and the specified condition holds. Statecharts were used successfully in the MENTOR project (Wodtke *et al.* 1997). An advantages of this approach is that implicit OR-split can be easily modeled (Van der Aalst 1998).

Petri net based models are a good example of the third approach, and are also used in variety of workflow products like INCOME/STAR (Promatis GmbH) or COSA (Software-Ley GmbH, (Sof 1998)). Petri nets were also chosen as a workflow model in the Bond system.

The transitions between states in the Bond system are unconditional and cannot directly cause any actions to be performed, the routing functionality has to be implemented in the strategies. Therefore the strategies used in Bond for workflow management belong to one of the two categories:

- domain strategies that implement domain functionality and are followed by only one transition called "success"
- routing strategies that decide which activities to perform next and implementing synchronization

The set of states in the state machine of a workflow can be divided into two disjoint subsets depending upon the type of strategy performed. There are no transitions among states performing domain strategies or among states performing routing strategies. Thus the state diagram of the Bond agent implementing an workflow is an oriented, bipartite graph. If we add the requirement that the state associated with a domain strategy cannot be entered before all the preceding states associated with routing strategies are entered, and that after leaving the state associated with the domain strategy all the following states performing routing strategies are entered, it is clear that the execution of the Bond agent implementing an workflow can be modeled by a Petri Net. In this mapping the states of the Bond state machine performing domain activities correspond to Petri net transitions, the states performing routing activities correspond to places, and Bond transitions correspond to arcs. The markings of the Petri net correspond to the routing states the Bond agent is in at a given time.

Interoperability and Meta–Models

As a part of its standardization effort, the Workflow Management Coalition defined a common meta-model for describing the workflow process definition (see Fig. 1) and a textual format for the interchange of workflow process definitions – the Workflow Definition Language (WfDL) (Workflow Management Coalition 1998). Some authors believe that the semantics of workflows is still not well defined in WfMC standards, and Petri nets should be considered a lingua franca for workflows . The entities defined by the model WfDL include:

(a) workflow participants representing sets of resources or humans that can act as performers of various activities in the process definitions;

(b) workflow applications which may be invoked to support or wholly automate the processing associated with each activity;

(c) process activities representing logical, self-contained units of work to be processed by either participants or applications;

(d) transitions defining flow of control between activities with optional conditions associated with them; participant definitions; and

(e) workflow relevant data, information passed among activities that can influence the course of execution of the workflow process.

In this paper we shall be mostly concerned with the entities describing flow of control in the workflow process definition, that is activities and transitions. The design choice of the authors of the WfDL specifications was to include as much of the routing information as possible in the activity definition and retain the transition definitions as simple routing assignments. In effect each activity description has three sections,

(1) A "prologue" defining the behavior of the activity if multiple incoming transitions exist, referred to as the join type of the activity,

(2) The definition of the actual activity, and

(3) An "epilogue", that describes the behavior when multiple outgoing transitions exist, referred to as the split type of the activity. An activity A's split (join) type is `AND` if many activities following (preceding) A are to be performed in parallel; activity A's split (join)

type is `XOR` if only one of the specified activities is to be performed after (before) A.

Some authors believe that the semantics of workflows is still not well defined in WfMC standards, and Petri nets should be considered a lingua franca for workflows .

Bond Workflow Management Architecture

An objective of the workflow management framework proposed in this paper is to build case manager agents based upon a static description of an workflow. There are several ways of specifying a workflow. Several existing commercial products use a Petri-Net based definition language or design tool, while the Workflow Management Coalition defined the Workflow Definition Language as an industry standard (Workflow Management Coalition 1998). While we intend to base our framework on Petri nets concepts and possibly maintain interoperability with the above mentioned products, we would like to provide a translation facility from the Workflow Definition Language to the Petri net based representation. This representation requires further transformations to obtain a Bond blueprint. The following sections will present these procedures in detail.

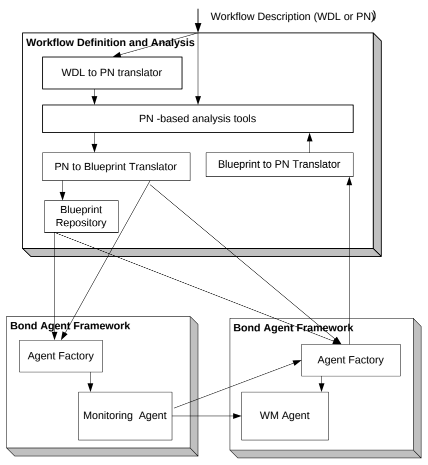

Figure 3: Workflow management in Bond

Figure 3 illustrates the definition and execution of a workflow in Bond. Our design supports dynamic workflows and complex monitoring. The workflow management agent originally created from a static description can be modified based upon the information providing by the monitoring agent. Several workflows may be created as a result of mutations suffered by the original workflow (Van der Aalst & Basten 1999). Once the new blueprint is created dynamically, it goes through the analysis procedure and only then it can be stored in the blueprint repository. The distinction between the monitoring agent and the workflow management agent is blurred, if necessary they can be merged together into a single agent.

Translation from the Workflow Definition Language to the Petri Net representation

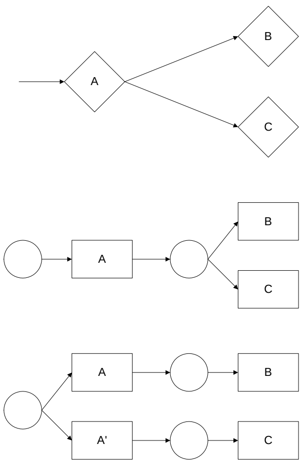

Figure 4: Handling of OR-splits. The top diagram presents an activity A connected to activities B and C by an OR-split. The diagrams below show how this fragment of a workflow definition can be translated to Petri nets

The basic objective while designing the translation procedure was to maintain close resemblance of the resulting Petri net to the original Workflow Definition Language input. Another requirement was that the resulting Petri net was meant to be used as a base for a Bond blueprint. In particular it is necessary to avoid duplicating transitions while translating OR-splits; for analysis purposes, as presented in (Van der Aalst 1998), OR-splits can be modeled by allowing the precondition place to be connected to multiple transitions corresponding to multiple copies of the original activities, each of them with only one postcondition (see bottom diagram in Figure 4). This is not acceptable for our purposes, because if we map Petri net nodes to strategies, it is necessary to make the actual routing decision *before* the strategy corresponding to the activity is performed. However this routing decision has to be made based on the outcome of execution of this strategy.

The translation algorithm proposed herein iterates over all the transitions defined in the Workflow Definition Language model file and classifies each of them based upon the routing information contained in the descriptions of the adjacent activities. Each transition can originate from an activity that has either one outgoing transition or multiple outgoing transitions forming either an AND split or an XOR split. Similarly, each transition can be followed by an activity that has either one incoming transition or multiple incoming transitions forming either an AND join or an XOR split. Thus there are 9 cases to be considered. They are depicted on the left hand side in Fig. 5 and for each of

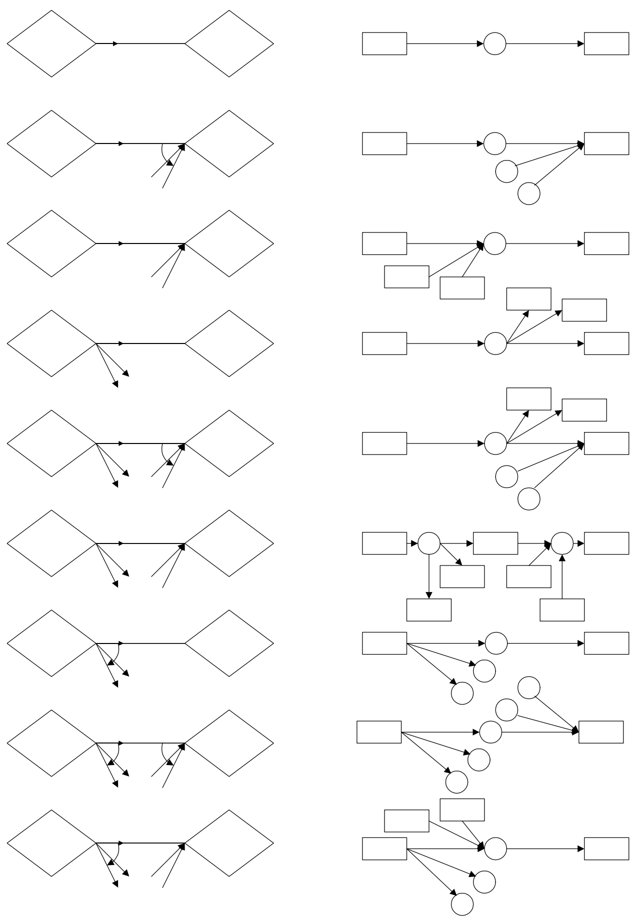

Figure 5: 9 cases arising in translation of the workflow description from the Workflow Definition Language to Petri net based representation. An arc spanning arrows connecting activities denotes an OR-split or join; absence thereof denotes an OR-split or join.

them the translation is shown on the right hand side.

The translation is straightforward. However, special care must be taken when a transition originates from some activity A with an OR split and leads to another activity D with an OR join. Activity A has one postcondition and activity D has one precondition but they cannot be merged because incorrect execution paths would appear in the resulting Petri net (see Fig. 6 for an illustration). Therefore a “dummy” Petri net transition is inserted to connect A’s postcondition place to D’s precondition place.

Note that the Workflow Definition Language specification provides for a special kind of activity (marked with the keyword `ROUTE`) to allow for more complex routing constructs – therefore the introduction of “dummy” transitions in the Petri net representation does not depart too far from the conceptual model of the Workflow Definition Language. Because the Petri net representation resembles closely the Workflow Definition Language representation, it is possible to translate back from the Petri net representation to the Workflow Definition Language representation. This is advantageous for interoperability reasons, especially because we would like to change the workflow definition dynamically, at execution time, and reflect that change into a new workflow expressed in the Workflow Definition Language.

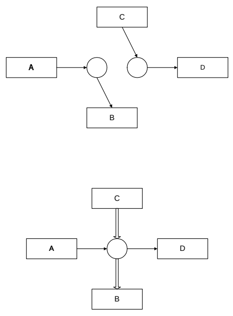

Figure 6: Spurious execution path `C => B` appears when a workflow transition connects an activity A with an OR-split with an activity D with an OR-join and A’s postcondition place is merged with D’s precondition place.

Translation from the Petri Net representation to Bond blueprint

The purpose of this section is to present a method of simulating a Petri net on the Bond multi–plane finite state machine that allows the Bond agent framework to function as a workflow enactment system. In this case the strategies for states corresponding to Petri net transitions are used to perform workflow activities. The approach presented here is not limited to the workflow management domain i. e. it can be viewed as an extension to the Bond system that allows Bond agent definitions to be formulated in the language of Petri nets.

The translation from the Petri Net representation to the Bond blueprint is most straightforward for a class of Petri nets called *S-nets.* In these Petri nets $| \bullet t | = 1 = | t \bullet |$ for every transition t, i. e. each transition has exactly one incoming arc and outgoing arc. If the initial marking of an S-net has exactly one token and the net is strongly connected then the net can contain only one token at any time (thus it immediately follows that it is safe) and in result it can be simulated by a single-plane Bond finite state machine.

Not all Petri nets are S-nets (which is quite fortunate in fact as their main use is to model concurrency absent in state machines), however for a large class of Petri nets (containing nearly all the nets useful for modeling workflows) there exists a systematic method of simulating them on the Bond multi-plane state machine. It is convenient to introduce here the following definitions and a theorem from (Desel & Esparza 1995).

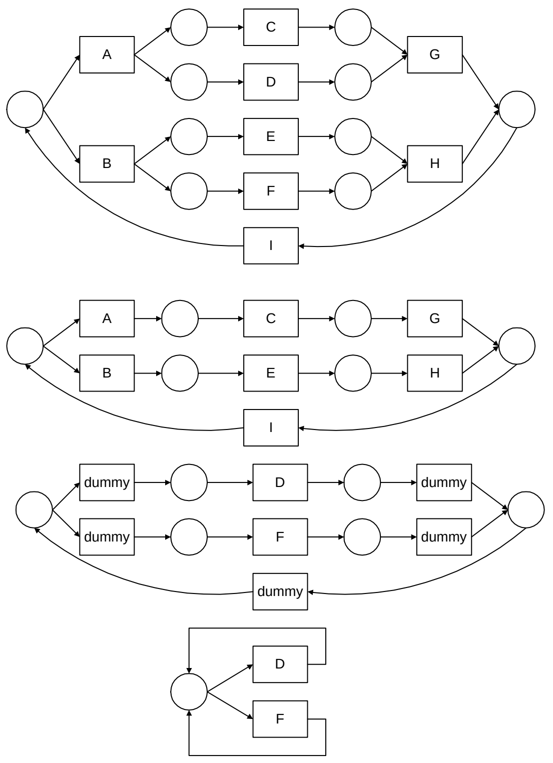

Figure 7: Example of the S-decomposition procedure. The top diagram represents the original net, the following two depict the two S-components with "dummy" transitions selected. The bottom diagram represents the second S-component after the dummy transitions have been contracted.

Subnets. Let (S, T, F) be a Petri net and $X \subseteq S \cup T$. The triple $(S \cap X, T \cap X, F \cup (X \times X))$ is the *subnet* of N generated by X.

S-components. Let N' be the subnet of a net N generated by a nonempty set X of nodes. N' is an *S-component* if

- $\bullet s \cup s \bullet \subseteq X$ for every place s of N
- N' is a strongly connected S-net.

S-covers. Let C be a set of S-components of a net. C is an *S-cover* if every place in the net belongs to an S-component of C. A net is covered by S-components if it has an *S-cover*.

Theorem Free choice, bounded and safe nets have S-covers.

Moreover, there exist efficient algorithms for finding S-covers of free choice, live and safe nets. The algorithm used in the current implementation is taken from (Lee, Nishimura, & Kumagai 1993) and runs in $O((|S||T|)^2)$.

This result constitutes the base of the translation procedure from Petri nets to Bond blueprints. We introduce a separate Bond plane for each of the S-components obtained from the decomposition procedure and populate it with Bond states corresponding to places (we shall refer to them as p-states) as well as transitions (further called t-states) in the S-component and add Bond transitions corresponding to arcs (see Fig. 7). However, the decomposition procedure does not produce disjoint components. Several Bond states may corresponding to one node in the original Petri net, in particular several t-states may correspond to one activity to be performed by the agent, which is clearly undesirable. To alleviate this problem we simply mark all but one of the t-states corresponding to one Petri net transition (which in turn corresponds to one activity) as "dummy" and associate with them a strategy that once started immediately terminates reporting success.

In order to be able to simulate the Petri nets on the Bond multi-plane state machine we also have to add synchronization functionality to the strategies assigned to p-states. This might seem to be a departure from the Petri net model where synchronization is explicitly expressed in the layout of the net, however, one has to keep in mind that in the Petri net model each transition possesses its own "thread of control" i. e. it can fire whenever it is enabled, independently of others, while in Bond a "thread of control" is assigned to each plane, and the Petri net transition cannot "decide" when it wants to fire because entering the state corresponding to a Petri net transition causes the associated activity to start executing. Even though it might be possible to implement a scheme in which the strategies associated with the states corresponding to Petri net transitions decided when to perform the actual work, this would break the design principle of separation of the routing and domain functionality.

Therefore in our design it is the role of the strategies associated with p-states to decide when to enter the t-state that follows them. Even though in general a transition can have many preceding places, each transition has just one preceding place in an S-net, and although a transition can belong to many S-components, it will be marked as "dummy" in all but one of them, thus the state in charge of synchronization can be determined unambiguously. The strategy for this state has to wait until all the other p-states corresponding to the places preceding the given transition are entered and succeed. Additionally it can wait until an (external) trigger condition necessary for firing of the transition holds.

Note that since the "dummy" transitions do not perform any useful work they can eventually be removed and their preceding and following places merged as long as no spurious execution paths are created and the routing conditions are correctly rewritten. This process can be iterated until no further improvements can be achieved (see Fig. 7 for an example).

Conclusions

In this paper we present an agent based workflow management system capable of supporting dynamic workflows. Data intensive applications of workflow management (e. g. climate modeling) require the dynamic modification of a workflow.

We describe briefly a multi-plane state machine agent model and its implementation. Agents are assembled dynamically from descriptions in the Blueprint language and can be modified while running.

We discuss problems pertinent to the translation of the Workflow Definition Language into Petri nets and of Petri nets into Bond blueprints. This project is part of the effort to develop an workflow management framework for the Bond system. The Bond system is available under an open license from `http://bond.cs.purdue.edu`.

Acknowledgments

The work reported in this paper was partially supported by a grant from the National Science Foundation, MCB-9527131, by the Scalable I/O Initiative, and by a grant from the Intel Corporation.

References

Alonzo, G.; Agrawal, D.; El-Abbadi, A.; and Mohan, C. 1997. Functionality and limitations of current workflow management systems. Technical report, IBM Almaden Research Center.

Bölöni, L., and Marinescu, D. C. 1999a. A Multi – plane State Machine Agent Model. Technical Report CS TR 99-027, Computer Sciences Department, Purdue University.

Bölöni, L., and Marinescu, D. C. 1999b. An Object-Oriented Framework for Building Collaborative Network Agents. In Kandel, A.; Hoffmann, K.; Mlynek, D.; and Teodorescu, N., eds., *Intelligent Systems and Interfaces.* Kluwer Publising House.

Bölöni, L., and Marinescu, D. C. 1999c. Biological metaphors in the design of complex software systems. *Journal of Future Computer Systems.* (in press).

Bölöni, L.; Hao, R.; Jun, K. K.; and Marinescu, D. C. 1999. Structural biology metaphors applied to the design of a distributed object system. In *Proc. Workshop on Biologically Inspired Solutions to Parallel Processing Problems*, LNCS, 275–283.

Chang, J. W., and Scott, C. T. 1996. Agent–based Workflow: TRP Support Environment (TSE). *Computer Networks and ISDN Systems* 28:1501.

Desel, J., and Esparza, J. 1995. *Free Choice Peri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press.

Harel, D.; Pnueli, A.; Schmidt, J. P.; and Sherman, R. 1987. On the Formal Semantics of Statecharts. In *2nd IEEE Symposium on Logic in Computer Science.*

Jun, K. K.; Bölöni, L.; Palacz, K.; and Marinescu, D. C. 1999. Agent–Based Resource Discovery. Technical report, Departament of Computer Sciences, Purdue University.

Lee, D.; Nishimura, T.; and Kumagai, S. 1993. Structural and behavioral analysis of state machine allocatable nets based on net decomposition. *IEICE Trans. Fundamentals* E79-A(3):399–408.

O'Brien, P. D., and Wiegand, M. E. 1998. Agent based process management: applying intelligent agents to workflow. *Knowledge Engineering Review* 13:2.

Software-LEY. 1998. *COSA Workflow 2.0 Product Specification.*

Tsompanopoulou, P.; Bölöni, L.; Marinescu, D. C.; and Rice, J. R. 1999. The Design of Software Agents for a Network of PDE Solvers. In *Workshop on Agent Technologies for High Performance Computing, Agents 99*, 57–68. IEEE Press.

Van der Aalst, W. M. P., and Basten, T. 1999. Inheritance of Workflows. An approach to tackling problems related to change. (draft).

Van der Aalst, W. M. P. 1998. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers* 8(1):21–66.

Wainer, J.; Weske, M.; Vossen, G.; and Medeiros, C. B. 1996. Scientific workflow systems. In *NSF Workshop on Workflow and Process Automation: State of the Art and Future Directions.*

Wodtke, D.; Weissenfels, J.; Weikum, G.; and Dittrich, A. K. 1997. The Mentor Project: Steps Towards Enterprise–Wide Workflow Management. In *Proceedings of the IEEE International Conference on Data Engineering.*

Worah, D., and Sheth, A. 1996. What Do Advanced Transaction Models Have to Offer for Workflows. In *Proceedings of the International Workshop on Advanced Transaction Models and Architectures (ATMA), Goa.*

Workflow Management Coalition. 1998. Interface 1: Process definition interchange process model. WfMC TC-1016-P v7.04.

Workflow Managment Coalition. 1996. Workflow Managment Coalition Terminology and Glossary. Technical Report WFMC-TC-1011, Workflow Managment Coalition.