A Comparison of Workflow Metamodels[*]

Yu Lei and Munindar P. Singh
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534, USA

ylei2@eos.ncsu.edu, singh@ncsu.edu

Abstract

Because of its immense promise in dealing with complex tasks in heterogeneous environments, workflow technology has drawn continuing interest from the research and commercial communities. Although much attention has (rightly) been focused on techniques for scheduling workflows and for interfacing them to legacy databases and applications, corresponding attention has not been directed toward their conceptual modeling. We believe that the success of conceptual modeling will determine the eventual success of workflow technology.

Every workflow system implicitly incorporates a metamodel. Metamodels are the objects of our study. We identify major categories of workflow metamodels, and a list of criteria to use as evaluation dimensions. We model fragments of a workflow in the manufacturing domain in the different metamodels, and compare their effectiveness in capturing the desired properties in a modular, reusable fashion.

This evaluation is useful not only in helping one decide which metamodel to employ, but also highlights some topics for future research.

[*]This research was supported by the National Science Foundation under grant IRI-9529179, IBM corporation, and the NCSU College of Engineering. We are indebted to Manny Aparicio for useful discussions.

Contents

1 Introduction

Briefly, workflows can be defined as composite tasks that comprise coordinated human and computational subtasks. At the very least, workflows separate the control and coordination aspects of the composite tasks from the details of the component tasks. They are frequently heterogeneous and distributed over multiple systems. This facilitates the update and reuse of workflows, and has led to much commercial and research interest in tools for constructing and managing them.

Much research attention has concentrated on techniques for workflow management, and not enough on techniques for workflow modeling. Every approach implicitly carries in it some representational language or *metamodel*, but the metamodels themselves have not been the objects of study and evaluation in recent research. Recognizing that modeling issues will become increasingly important as workflow technology attempts to expand its applications, we compare some of the leading approaches to workflow modeling. We do so in the context of a simple application from the manufacturing domain. This is a variation of the example scenario from the SHIIP (SHipbuilding Information Infrastructure Protocol) project being conducted by the NIIIP (National Industrial Information Infrastructure Protocol) consortium [SHIIP, 1997]. We are collaborating with one of the leading industrial members of SHIIP, so our interest is not exclusively academic.

The potential benefits of this exercise are twofold. From an engineering standpoint, an evaluation, by presenting the strengths and shortcomings of different metamodels, helps us choose a metamodel for our specific application. From a scientific standpoint, it highlights the opportunities for further research.

The rest of this paper is organized as follows. Section 2 lays out our key terminology, identifies four categories of metamodels and proposes a list of evaluation dimensions. Section 3 describes the SHIIP example with a detailed scenario. Section 4 introduces several representative metamodels and renders part of SHIIP representations in those metamodels. Section 5 carries out an overall evaluation for those metamodels. Section 6 offers concluding remarks.

2 Workflow Metamodels

2.1 Background Concepts

Some basic terms are used extensively in the literature. We present a short description for some important concepts. Although they are not formal definitions, these descriptions help clarify the meanings we adhere to in this paper. Figure 1 illustrates the relationship of those concepts.:

- A *task* is a definite piece of work.
- An *actor* is a human being or a machine which can perform a task.
- A *role* is a logical abstraction of one or more physical actors, usually in terms of functionality.
- A *process* is a business process, which is composed of tasks structured in an appropriate manner.
- A *process model* is an abstraction of business processes. It emphasizes the coordination of tasks by highlighting their interdependence.
- An *organization* aggregates actors into groups, which are structured in some way.

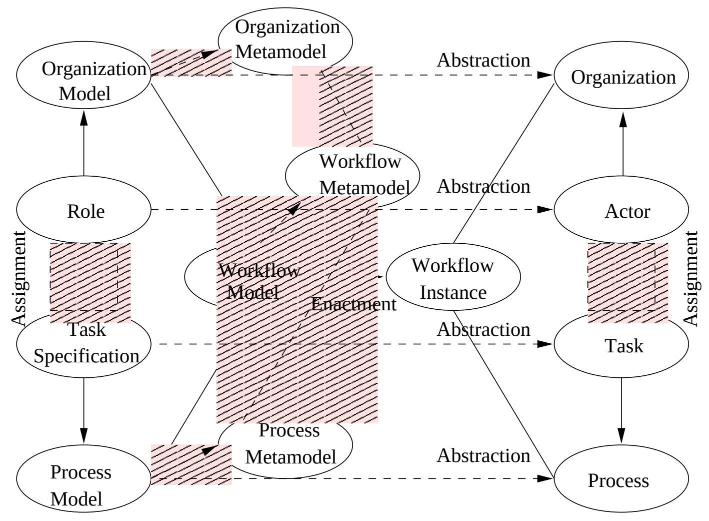

Figure 1: Workflow Concepts

- An *organizational model* is an abstraction of organizations.
- A *workflow (instance)* is a process combined with an organization, and assigns the tasks in the process to actors in the organization.
- A *workflow model* combines a process model and an organizational model.
- A *workflow metamodel* is a representational language in which to express workflow models.

2.2 Major Types of Workflow Metamodels

Since the process model is central to the workflow model, we identify four categories of workflow metamodel based primarily on their process metamodels.

Task flow Task is the fundamental element in the process metamodel of task-flow based workflow metamodel. It emphasizes the analysis of all the potential sequences of tasks in the specification phase. During model execution, an instance of the workflow decides which sequence to follow on the basis of static specification and dynamic running context. In the graphical representation of process model, typically a directed acyclic graph (DAG), tasks show up as the nodes, state information as conditions attached to the edges.

State transition A state abstracts the context information at a particular time when a system is running. A state transition occurs when a particular event or activity happens. The behaviors of a workflow are represented as a sequence of state transitions. In the process model DAG, states show up as a node, and tasks or events as conditions attached to an edge.

Relationship capturing Relationship-capturing based metamodel is founded at the view that the workflow brings all the tasks together on the basis of certain relationship. The process model is built by means of capturing this particular relationship. Example relationships can be triggers

[Joosten, 1994] or enabling or disabling conditions. The elements are typically tasks, so in certain respects these resemble the task flow metamodels.

Communication based These approaches identify the communications among the various actors. They must also involve tasks, of course, but they present the tasks as occurring in the context of the relationships and communications of the actors, not independently.

2.3 Key Dimensions of Metamodels

We propose a list of evaluation dimensions for workflow metamodels. We omit concerns such as ease of use and compatibility with programming languages that are not related to metamodels. The abbreviations in parentheses are used in Table 1.

Granularity Granularity indicates the abstraction level of the basic element examined by a metamodel. A task represents a meaningful business action that is. A task might have a fine structure among its constituent tasks [Leymann & Roller, 1994]. Coarse granularity simplifies specification and increases flexibility, whereas fine granularity simplifies execution.

Control Flow (Control) Most metamodels support the basic (sequence, branch, and loop) structures in order to be programatically complete. Some add constructs for synchronization, parallel programming (e.g., fork and join). Some also add constructs to better support concurrency [Chan & Vonk, 1997].

Data Flow (Data) Data flow represents potential paths followed by information exchange. Some metamodels provide a separate representation for data flow; others merge it into control flow with control flow; still others merge it but annotate it explicitly.

Organizational Model (Org) An organizational model consists of three parts: role description, role relationships or groups, and an actor assignment policy [Bußler, 1994]. A model should provide the users with the capability to customize actor types. The structure of role groups can be described in terms of relational, hierarchical, and network models. Assignment policy accomplishes the mapping between logical roles and physical actors.

Role Binding (Rol) Role binding is an important technique to enable dynamic actor assignment by loosening the coupling between roles and actors. An actor can adopt the responsibility of a role by binding corresponding run-time support. During execution, an actor can simultaneously claim different roles; a role can simultaneously be embodied by different actors. Role binding sometime includes support for role delegation, which betters the load balance by means of necessarily delegating part of responsibilities taken by a busy role to others.

Exception Handling (Exc) An exception refers to the situation where unexpected behaviors occur during the system execution. Exception handling concerns how to respond properly to exceptions when preserving the legality of system states in case any exception happens. There are two major kinds of exceptions in workflow systems: execution exception and organizational exception.

An execution exception occurs when the system runs out of normal control flow; an organizational exception is caused by illegal changes in the organizational structure. A common structure of the specification of exceptions is made of a condition-reaction pair: when the condition is verified, the corresponding reaction is executed [Casati *et al.*, 1997]. Another rule-based formulation uses nonmonotonic reasoning to handle exceptions [Singh & Huhns, 1994].

Transaction Support (Trn) A transaction is defined as a set of operations that satisfy ACID properties, namely, atomicity, consistency, isolation, and durability [Gray & Reuter, 1993]. It is crucial to enforce the satisfaction of ACID properties for all the transactions in order to preserve consistency all the time. However, traditional transaction support, like 2-phase lock protocol, are not suitable for heterogeneous environments. Despite this, a good workflow metamodel should provide some support for structuring tasks and guaranteeing some of the properties of their joint execution.

Commitment Support (Com) Several subtle forms of commitment can arise in workflows. These can be specific to different actors, and reflect the organizational structure of the workflow. An actor may commit to another to notify him of any changes he effects, without committing to inform everybody. The challenge is to build new support mechanism to attain more flexibility compared to traditional support while maintaining the robustness of workflow systems.

3 SHIIP Example

One of the SHIIP scenario involves a shipyard *Foreman* requesting a configuration group to reconfigure or upgrade a prefabricated combinational part [SHIIP, 1997]. We adapt this scenario to better support our goal of comparing different approaches.

We define the scenario as follows:

1. *Foreman* fills out a report for reconfiguration and submits to *Configuration Group*
2. *Manager* of *Configuration Group* checks the report. *Manager* may accept or reject the report. For instance, if she finds that the problem could be fixed by means of modifying the old installation procedure, *Manager* would return the report to *Foreman* with helpful comments.
3. If the report is returned, the process terminates. Otherwise, it proceeds.
4. *Manager* makes a financial budget to estimate the cost of performing the reconfiguration. After the budget is forwarded to *Account Office*, *Manager* awaits the decision from *Account Office*.
5. If the budget fails to pass in *Account Office*, the process terminates. Otherwise, it proceeds.
6. *Manager* delegates a *Design Team* to work out a design scheme for the requested reconfiguration.
7. *Manager*, along with *Foreman*, checks the scheme from *Design Team*. If the scheme is not satisfactory, it is returned to *Design Team* with helpful comments for redesign. Otherwise, it proceeds.

8. *Manager* sends out the approved scheme to *Fabricator* asking for fabrication.

9. *Fabricator* identify the parts needed to carry out the fabrication. For those parts unavailable in stock, *Fabricator* sends out a order form to *Supplier*.

10. *Fabricator* proceeds to fabricate when all the parts are available. The finished combinational part is forwarded to *Configuration Group*.

11. *Manager*, along with *Design Team*, checks the fabricated part from *Fabricator*. If the part exactly embodies their design scheme, the part is delivered to *Foreman*. Otherwise, the part is returned to *Fabricator* with helpful comments to request refabrication.

12. *Foreman* accepts the reconfigured part and applies it to the project he is working on.

The above scenario describes the normal execution sequence. However, some unexpected situations might happen in the real world. For the purpose of illustration, we consider three exceptions:

- An exception will occur when a *Manager* actor does not exist to receive the request from *Foreman*.

- *Manager* checks the format of the report when he receives a reconfiguration request from *Foreman*. If some required items—e.g., the part number to be reconfigured—are missing, an exception procedure would be initiated to ask for a corrected report.

- When the parts delivered by the *Supplier* are not what *Fabricator* expected, an exception procedure would be initiated to have the parts exchanged for correct ones.

The dividing line between normal execution and exceptions is not absolute. We can integrate an exception handler as a part of normal sequence. For instance, we can add the procedure as a routine for *Fabricator* to check the parts from *Supplier* and request a resupply if necessary. However, the separation of handling unexpected situations, which do not happen often, helps us to focus on the central control flow and modularize requirement capture.

4 Applying the Metamodels

We choose several representative metamodels as the subjects with which we proceed our evaluation. These metamodels cover the categories described in section 2.2 and are well-known. We introduce each metamodel briefly. To save space, we only represent snippets of the above scenario—mostly about the design subprocess of step 7—in each metamodel.

4.1 FlowMark

FlowMark is a product developed by IBM corporation [Leymann & Roller, 1994]. The metamodel of FlowMark is task-flow based. It includes the following syntactic elements: *activity*, *control connector*, *transition condition*, *container*, *data connector*, *exit condition*, *synchronization condition*, and *task*. In the graphical representation, *activities* are basic nodes. *Control connectors* are directed edges which bind tasks together to represent the control flow. *Transition conditions* are boolean

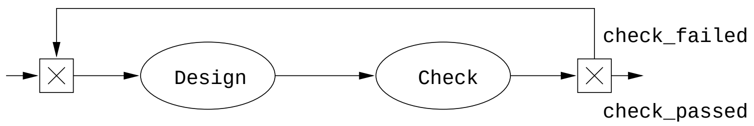

Figure 2: FlowMark

expressions associated with control connectors. Only control connectors with true transition conditions are followed at run-time. *Containers* are attached to tasks as the holders of input and output parameters. Like control connectors, *data connectors* make links between containers to indicate the data flow. Since the termination of a task does not entail that it has been finished successfully, because it may have an unexpected interrupt. Therefore, the *exit condition* records the success or failure of a task. *Synchronization conditions* specify when the execution can continue in case that several connectors point to the same target. Coupling each activity with an actor results in a pair called *task*. The design subprocess is illustrated in Figure 2.

4.2 WIDE

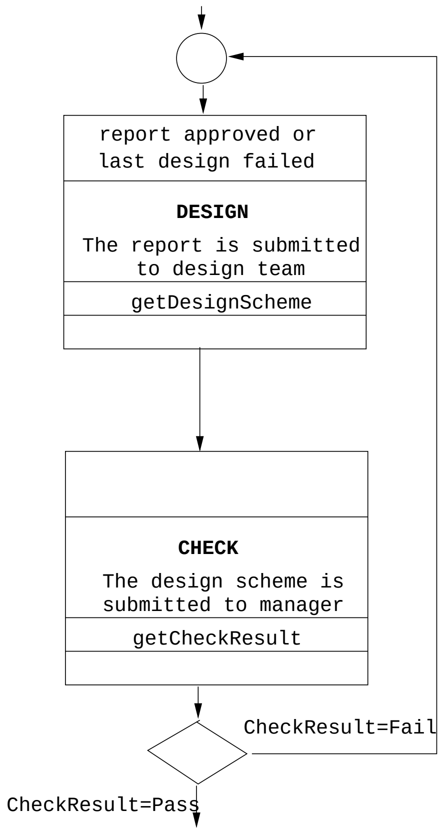

Figure 3: WIDE

WIDE (Workflow on Intelligent and Distributed database Environment) is an ESPRIT project of the European Commission [Casati *et al.*, 1997]. The WIDE metamodel is task-flow based. Each task is specified by the following components: *precondition*, *action*, *postcondition*, *role constraint*, and *exception handler*. *precondition* and *postcondition* have to be satisfied when the task starts and ends. *Role constraint* expresses the task assignment policy that declares the qualification of the roles that can perform this task. Besides the support for a strong organizational model, WIDE model enables dynamical role-binding. *Exception handler* includes two parts: *condition* and *reaction*. The reaction part is activated when the condition part are satisfied. Efficient exception handling has been achieved with active rules. A number of connectors— total fork, total join, conditional fork, and partial join—are provided to represent the potential control sequences of tasks. Besides, from the perspective of methodology, WIDE emphasizes the integration with external databases and provides strong transaction support by the coordination between a global and multiple local transaction managers. The design subprocess is illustrated in Figure 3.

4.3 Statecharts

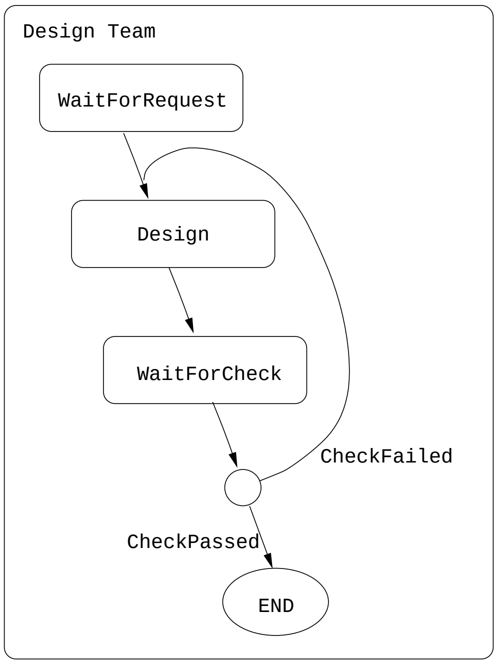

Figure 4: Statechart

Statecharts are a typical state-transition based metamodel [Harel & Gery, 1997]. A statechart is a traditional state transition diagram with three extensions of hierarchy, concurrency, and communication between concurrent tasks. Hierarchy facilitates modular development by means of capturing a system at different levels of abstraction. The support of concurrency is crucial for modeling most reactive systems. Cooperative concurrent tasks work together by exchanging information with each other. A system starts from a start state and terminates at a final state. State transitions are associated with events or tasks, where a transition happens as the result of the occurrence of a task.

Following state transition paths, we can get potential sequences of tasks by means of linking tasks associated with state transitions together. The design subprocess is illustrated in Figure 4.

4.4 Trigger Metamodel

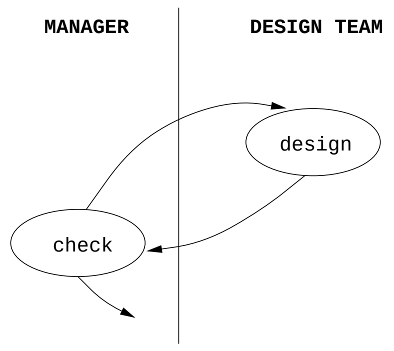

Figure 5: Triggers

The Trigger metamodel is relationship capturing [Joosten, 1994]. A task is defined as a set of events that occur under the responsibility of one actor. This definition is important to represent those tasks that are otherwise considered to be performed by multiple cooperative actors. An event triggers a task if the occurrence of the event causes the task to be performed. A workflow is defined as a system whose elements are tasks, related to one another by a trigger relation, and triggered by external events. The trigger metamodel can distribute one trigger over several others, or synchronize several triggers into one. The trigger metamodel is easily translated to a petri-net model, and thus it can be readily used for workflow automation. A separate communication model is provided to capture the data flow structure. The design subprocess is illustrated in Figure 5.

4.5 ActionWorkflow

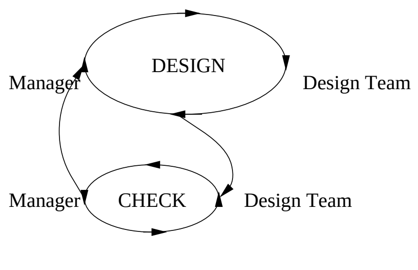

Figure 6: ActionWorkflow

ActionWorkflow is a product developed by ActionTech [Medina-Mora *et al.*, 1992; Winograd, 1988; Denning, 1992]. In its metamodel, any task is regarded as a closed loop composed of four phases: *proposal*, *agreement*, *performance*, and *satisfaction*. There is always an identified *customer* and *performer*. The performer completes actions leading to the satisfaction of the customer by going through the following phases: (a) the customer makes a request for some work to be done; (b) the customer and the performer reach an agreement on the work specification; (c) the performer takes actions to complete the work; and (d) the customer accepts the completion with satisfaction. Three kinds of relationships are identified among task loops. A loop might be part of another loop, trigger another loop, or resolve another loop. In other words, a task loop may complete a part of work of another loop, cause the initiation of another loop or help make the decision as to which action to take in another loop. ActionWorkflow provides three connectors—*conditional*, *splitter*, and *rendezvous*—to refine the trigger relationship. The first enables the branch trigger. A branch is taken only if the associated condition is true. The latter two connectors are used to deal with the concurrent trigger. We use a splitter when a task loop triggers multiple other loops at a time and a rendezvous when multiple loops trigger one loop together. The design subprocess is illustrated in Figure 6.

5 Evaluation

Different metamodels are constructed from different perspectives. It would not be accurate to assert that one approach is superior to another in all circumstances. However, there are always some trade-offs. A metamodel might miss some aspects when it concentrates on others. The purpose of our evaluation is to highlight those trade-offs.

We first make an evaluation at the level of major types of metamodels. Among the four types of metamodels, task-flow based metamodel is the most straightforward in the sense that people make a schedule in about the same way. Since they focus on how to bring tasks into order with little concern for their location, almost all of the task-flow based metamodels have centralized schedules.

In contrast, state-transition based metamodels lead to a distributed control logic. Although we can consider the state transitions from the perspective of the whole system, most state-transition based metamodels have individual state-transition diagrams for each role, which can communicate with each other as necessary. In a sense, such metamodels examine the system with a distributed viewpoint.

Relationship-capturing metamodels propose a novel perspective to represent a workflow system. However, such metamodels are built on the basic assumption that the system behaviors can be fully captured by certain relationship. The assumption is somewhat restrictive, This might explain why only few of such metamodels are available.

Communication-based metamodels offer an interesting approach for workflows. These are especially suited to groupware settings, and not so much to data-intensive applications. They have the advantage of identifying the actors for each task, but this can also be a disadvantage when multiactor tasks are considered,

Now we turn to the metamodels used in our examples. Table 1 summarizes the features of these metamodels in terms of the dimensions introduced in Section 2.3. In this table, *together* indicates that data flow is represented in the same model as control flow, but with different markings for distinction; *separate* means that data flow is modeled separately; *implicit* means that there is no explicit data flow representation. Note that implicit data flow can be reasonable, since data exchange may be accomplished by communications carried out for control flow. However, explicit data flow is clearer. The granularity of Statechart and Triggers is event, because transitions and triggers can be thought as forms of events. To support synchronization, ordinary branching operators are often extended by, for instance, introducing exclusive OR/AND join operators in FlowMark and WIDE. WIDE further introduces a multitask operator for concurrency support [Chan & Vonk, 1997]. FlowMark and WIDE both support an organizational model. WIDE has achieved greater

Approach	Granularity	Control	Data	Org	Rol	Exc	Trn	Com
FlowMark	task	basic, synchronization	together	yes	no	yes	no	no
Action	task	basic, synchronization	implicit	no	—	yes	no	yes
WIDE	task	basic, synchronization, concurrency	separate	yes	yes	yes	yes	no
Statecharts	event, task	basic	implicit	no	—	no	no	no
Triggers	event, task	basic	separate	no	—	no	no	no

Table 1: Evaluation Dimensions for Different Metamodels

flexibility with the dynamic task assignment supported by role-binding. ActionWorkflow does not support organizational exception handling because it does not have an organizational model at all. However, ActionWorkflow provides some rudimentary commitment support. Transaction and commitment support receive little attention in current approaches, with the exception of WIDE, which provides transaction support through the cooperation between a global transaction manager and local transaction managers.

Let us consider the approaches in more detail. First, the task-flow based metamodels, FlowMark and WIDE, bear much similarity from the viewpoint of conceptual modeling. Both models take tasks as atomic. They schedule tasks by capturing their temporal interdependence and then build an organizational model to assign those tasks to actors for organizational constraints. In contrast, ActionWorkflow considers task as divisible. It captures the static task hierarchy by the relationship that one task loop could be part of another one. Also, a task is always associated with a requester and performer.

Second, the state-transition based metamodel, Statecharts face a central issue about how to define the states. Too fine a definition leads to a large number of states, which might be intractable in some cases; too coarse to a composite activation part, where Statecharts do not provide an efficient mechanism to describe the internal structure of the activation. Also, although Statecharts allow the communication between two state diagrams, little concern has been paid to mechanisms to support such communication.

Third, the Trigger metamodel is a perfect example of relationship-capturing metamodel. The correspondence between the Trigger metamodel and Petri-nets contributes the most significant feature, which enables workflow automation and distributed control logic. However, the Trigger metamodel requires identifying a responsible role for any task. This requirement is not suitable for some cooperative tasks because such cooperative information is lost in the representation of the Trigger metamodel. For our example, the check procedure looks like a task of *Manager* in the graphical representation of Trigger metamodel, even though it is performed by *Manager* and *Foreman* together in our scenario. Also, the Trigger metamodel does not provide any conditional branch, which does not seem reasonable for the practical application.

Fourth, the construction of models in ActionWorkflow is more complicated than in FlowMark or WIDE. However, more information is captured in ActionWorkflow. On the one hand, the coordination behavior between the requester and performer represented in ActionWorkflow model facilitates the support of commitment. On the other hand, the emphasis on the coordination of tasks causes some problems for ActionWorkflow. It does not appear appropriate to name a requester and performer for local tasks. Another problem is that the sequential control relationships are not well-defined among the components of a composite task.

6 Conclusions

Although we considered only a few products and projects, the ideas cover many of the cases of interest. A large number of workflow products exist, but they are not based on radically differing metamodels. Some of the metamodels already have associated enactment tools; such tools are under development for some of the others. This is encouraging, since it is only by enacting the workflow models can we be sure that the desired behavior is being obtained.

The various models have strong similarities in how they handle tasks and control flow among them. However, they differ the most in advanced features such as concurrency, role-binding, and transaction and commitment support. In fact, their coverage of these features still leaves much to be desired. This is probably because the workflow research community still has not come to a good understanding of what these issues entail. However, these limitations have hindered the deployment of workflow technology in managing tasks in large-scale, open, heterogeneous enterprises.

We expect that the near future will witness the following developments. First, more formal methodologies will be developed. We believe that metamodels ought always to be accompanied by methodologies for how to use them in practice. However, presently, the methodologies for workflow metamodels have not been very well worked out. Methodologies will seek closer integration with external database schemas [Casati *et al.*, 1997]. Databases provides excellent persistent storage and strong local transaction support. The open issues for such integration include how to efficiently access external, typically heterogeneous, databases and how to make the extension of global transaction and commitment support.

Second, metamodels will facilitate naturally decentralized models and environments. This contrasts with extant metamodels, which typically presuppose a centralized control structure. In general, there are two alternatives to achieve distributed solutions. One is to use distributed programming technology, like COBRA, to make multiple servers transparent from users. Another way is to remove client/server architecture by means of distributing the control logic itself. The second, more radical, approach requires more expressive metamodels in which to capture the distributed control aspects.

Third, we envisage the widespread application of the agent metaphor. Agents provide a major step beyond distributed control. This is the use of higher-level abstraction. One such abstraction is commitments—more advanced than ActionWorkflow's—which we have begun for use in a potentially more powerful approach to workflows [Singh, 1997; Jain & Singh, 1997].

References

[Bußler, 1994] Bußler, C. J.; 1994. Policy resolution in workflow management systems. *Digital Technical Journal* 6(4):26–49.

[Casati *et al.*, 1997] Casati, F.; Grefen, P.; Pernici, B.; Pozzi, G.; and Sánchez, G.; 1997. WIDE workflow model and architecture. http://www.sema.es/projects/WIDE/Documents/.

[Chan & Vonk, 1997] Chan, Daniel K. C. and Vonk, Jochem; 1997. A specification language for the WIDE worklow model. http://www.sema.es/projects/WIDE/Documents/.

[Denning, 1992] Denning, Peter J.; 1992. Work is a closed-loop process. *American Scientist* 80:314–317.

[Gray & Reuter, 1993] Gray, Jim and Reuter, Andreas; 1993. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, San Mateo.

[Harel & Gery, 1997] Harel, David and Gery, Eran; 1997. Executable object modeling with statecharts. *IEEE Computer* 31–42.

[Jain & Singh, 1997] Jain, Anuj K. and Singh, Munindar P.; 1997. Using spheres of commitment to support virtual enterprises. In *Proceedings of the 4th ISPE International Conference on Concurrent Engineering: Research and Applications (CE)*. International Society for Productivity Enhancements (ISPE).

[Joosten, 1994] Joosten, Stef; 1994. Trigger modelling for workflow analysis. In *Proceedings of CON: Workflow Management*, Munich. R. Oldenbourg Verlag.

[Leymann & Roller, 1994] Leymann, Frank and Roller, Dieter; 1994. Business process management with flowmark. In *Proc. of COMPCON Spring 1994*. IEEE.

[Medina-Mora *et al.*, 1992] Medina-Mora, Raul; Winograd, Terry; Flores, Rodrigo; and Flores, Fernando; 1992. The Action Workflow approach to workflow management technology. In *ACM, Proceedings of the Conference On Computer-Supported Coooperative Work*.

[SHIIP, 1997] About SHIIP. http://shiip.npo.org/about-SHIIP.html.

[Singh & Huhns, 1994] Singh, Munindar P. and Huhns, Michael N.; 1994. Automating workflows for service provisioning: Integrating AI and database technologies. *IEEE Expert* 9(5):19–23.

[Singh, 1997] Singh, Munindar P.; 1997. Commitments among autonomous agents in information-rich environments. In *Proceedings of the 8th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW)*. 141–155.

[Winograd, 1988] Winograd, Terry; 1988. A language/action perspective on the design of cooperative work. *Human-Computer Interaction* 3(3–30).