A META LEVEL ARCHITECTURE FOR WORKFLOW MANAGEMENT

C. Beckstein and J. Klausner
Institut für Informatik, Universität Jena, Germany

Current workflow management systems still lack some important features, esp. intelligent failure handling mechanisms. This is due to the static and only implicitly defined meta model of these systems. We argue that intelligent failure handling needs semantically enriched workflow models. We, therefore, propose a meta level architecture for WFMSs providing an explicit and dynamic meta model. This complex architecture is supported by an intelligent assistant based on established AI techniques. The system uses a plan representation for workflow models. With this representation the system is not only able to do intelligent failure handling, but can also support the workflow modelling and execution processes.

1. Introduction

Workflow management systems (WFMSs) coordinate the automated execution of business processes (workflows) and have already become a key technology in numerous application domains. The WFMS coordinates the activities belonging to a business process on the basis of a formal representation of the workflow. This formal model is called a workflow model.

Figure 1 shows the graphical representation of an example workflow model „trip preparation“. This workflow consists of four activities. Three of them are primitive activities and can directly be mapped to user tasks or applications. The fourth („reservations“) is a complex activity (a subworkflow), which itself can be represented as a workflow model. Figure 1 also shows the control flow (arrows) and data flow (dashed arrows) of the workflow. Some other important aspects of a workflow model are not shown (e.g. the hierarchical structure of the organization consisting of supervisors, employees, groups, group leaders).

Figure 2 illustrates the traditional way of workflow modelling. Syntax and semantics of a workflow model are defined by a so called workflow meta model covering all important aspects of a workflow. In today’s (commercial) systems this meta model is fixed and only implicitly defined. Part of every WFMS is a buildtime client which is used to define workflow models (e.g. by means of a graphical editor). Once a workflow model is complete it is saved into a repository and can then be used as a complex activity in other (super)workflows. At runtime the user can instantiate workflow models as required by the business processes to be executed. These workflow instances are then executed by the so called workflow engine.

In order to get a feeling about the disadvantages of this traditional architecture we will now take a closer look at the failure handling mechanisms of WFMSs.
Fig. 1 An example workflow model.

Fig. 2 Traditional workflow modelling.
2. The Case of Failure Handling

The simple abortion of a crucial workflow in the presence of failures can lead to significant (financial) disadvantages for an organization and should therefore be avoided whenever possible. Hence, a WMFS needs flexible mechanisms that deal with such failures and exceptions and guarantee the consistent and reliable execution of workflows. Failure handling in commercial WFMSs is nevertheless mostly limited to mechanisms that just ensure the recovery of persistent meta data after system failures (e.g. a system crash). We are mainly interested in semantic failures: failures within the modelled business process (e.g. the flight reservation fails). Automated handling of such failures is not yet sufficiently supported. Currently, failure handling routines have to be explicitly specified within the workflow model. This only permits a very rigid handling of expected failures. Furthermore, the workflow designer alone is responsible

1. for the decision whether a failure handling routine must be provided at all and
2. for the correctness of the failure handling routine with respect to the modelled business process.

2.1. Enhanced workflow meta models

In view of the obvious disadvantages of explicitly specified failure handling strategies some researchers have proposed the application of advanced transaction models to workflow activities. The workflow meta model WAMO (Eder and Liebhart, 1995), e.g., is based on the flexible transaction model. It allows for the specification of alternative execution paths. The system can automatically choose among two default failure handling strategies depending on the properties of a failed activity: the simple repetition of the failed activity or the compensation of some successful tasks and the continuation with an alternative execution path. Figure 3 shows two workflow models specifying the same failure handling strategy. In the first one the failure handling routine was modelled explicitly. The second one uses WAMO’s meta model leading to a much more compact representation.

Another example is Leymann’s extension of the FlowMark meta model, called spheres of joint compensation (Leymann, 1995). These spheres are sets of workflow activities which have to be compensated all together in case one of them fails. They define static dependencies between the activities of a sphere. For an overview about transaction based workflow recovery please refer to (Klausner and Beckstein, 1997).

Each of these workflow meta models can only partially solve the problems with semantic failures. They just offer a slightly enhanced workflow meta model (Fig. 4) with a limited and fixed set of rigid (although adjustable) default strategies for failure handling. Among those strategies one is chosen automatically in case of a semantic failure. But this choice is based solely on static meta data (part of the workflow model) and does not depend on the actual cause of the failure. As the next example will show, this is often not appropriate.

2.2. Good failure handling requires rich semantics

Let’s assume our workflow model uses the WAMO meta model and contains three alternative execution paths (Fig. 5). If the first one fails, the previously completed activities back to the last choice point must be compensated for and the next path be chosen. If this path also fails, one has to go back again and try the third alternative which will then hopefully succeed. This strategy can be described as chronological backtracking with a fixed chronology.

This strategy has several disadvantages:

1. Compensation is costly. There is a lot of unnecessary work that has to be done.
Fig. 3 Explicit modelling vs. WAMO.

Fig. 4 Enhanced workflow meta models.

Fig. 5 WAMO's failure handling strategy: chronological backtracking.
2. Many workflow activities simply are not compensatable (e.g. drilling a hole).
3. Compensating activities must not fail.
4. Failure handling is done independent of the actual cause of the failure.

If the system would have more information about the inter-task dependencies a better failure handling strategy would be applicable. Let’s assume the first alternative fails because activity A needs a document which does not exist. If the system knew that activity B should have provided this document and that the executability of C also depends on the presence of this document, then it could avoid the (foredoomed) second alternative and continue right away with the third alternative path. This situation is shown in Figure 6.

Even this solution still requires the execution of compensating activities. This could be avoided if, in case of a failure, the system can reason about the current situation and search for a new execution path that will achieve the goal of the workflow. In our example this could involve the reexecution of activity B (Fig. 7). This strategy uses knowledge based workflow repair and could be called forward recovery. Of course, in order to use this strategy the system needs much more information about the semantics of a workflow than what is available in traditional workflow models.

3. A New and Enhanced WFMS-Meta Level Architecture

In the last section we argued that a WFMS needs lots of semantic information (meta data) about a workflow in order to reasonably handle failures. How can we supply the system with this information? We believe that with a fixed workflow meta model this is not possible. A fixed meta model can only support a fixed set of meta data and there could always be a situation where this is not sufficient. Instead, we propose to build WFMSs according to a meta level architecture (Maes and Nardi, 1988) discriminating between an object level and a meta level (Fig. 8).

The workflow modelling and workflow execution processes are running on the object level. The meta level defines an explicitly represented and dynamic meta model that provides the semantics for the object level processes. Additionally, this level runs processes that supervise and control the object level processes.

3.1. Features of the new architecture:

Compared with traditional WMFS architectures, our architecture has the following unique features:

1. explicitly represented meta model
2. dynamic meta data
3. intelligent assistant

The user has unlimited access to the explicitly represented meta model and can even define a part of it herself. She can, e.g., define her own failure handling strategies at the meta level and then use them in her workflow models later on.

An intelligent assistant serves as a mediator between the object level and the meta level. It automatically collects dynamic meta data relevant for the object level processes. Based on inferences about this meta data, it can dynamically adjust the meta level processes to the specific needs of the current situation. This also includes the automatic selection of a reasonable failure handling strategy for the type of a semantic failure under consideration.

An intelligent assistant provides a set of support tools for the object level processes as an interface to the meta level. At buildtime it can make suggestions to the workflow modeller how to improve/complete the workflow model he is currently working with. The intelligent assistant also provides tools...
Fig. 6 Backward recovery with forward checking.

Fig. 7 Forward recovery.
which support the exploration of workflow model changes and give information to the user about the impacts his changes would have on other parts of the workflow model ( “What if” - facility). Other important features provided at buildtime are validation and consistency checking.

At runtime the intelligent assistant supervises and controls the workflow engine and adjusts its execution model according to the stored dynamic meta data. Additionally, the intelligent assistant gives feedback to the user (e.g. a workflow administrator) about the ongoing workflow execution processes. It generates explanations and in case of semantic failures it informs the user about the failure type, failure source, and impacts of the failure. It can even make suggestions of how to successfully continue the workflow.

Let us assume, that in the example described in Figure 7 the failure has just been detected. In this case the intelligent assistant analyses the workflow specification and the dynamic meta data. This will result in the detection of the failure type - a missing input document. The intelligent assistant then further investigates the current situation. In this example it will recognize that compensation would be too time consuming and workflow repair should be preferred. The intelligent assistant then checks the inter-task dependencies between the activities. It will thereby identify which activity should have provided the missing document. With this knowledge it can then try to repair the workflow (e.g. by reinsertion of this activity). If this is successful, the intelligent assistant will present the fixed workflow to the workflow administrator. The user can then simply accept this suggestion or propose another solution. Note that this interactive workflow repair dialogue is directly supported by the buildtime module. Thus, our system tightly integrates the buildtime and runtime modules.

3.2. A closer look at the intelligent assistant

As the above scenario shows, the intelligent assistant has to combine several often orthogonal features. To achieve reusability and domain independence we suggest that it has to be built as a toolbox of coordinated tools that use various artificial intelligence (AI) techniques. These include AI planning techniques, RMS technology (Beckstein, 1996), and logic programming.

The main module of the intelligent assistant is a planner. At runtime, the planner is responsible for replanning in case of a failure. At buildtime it is used as an interactive tool that assists the workflow designer while creating new workflow models. Given a workflow specification (i.e. a set of activities together with a selection of constraints) this planner must be able to

- suggest a legal partial order of the activities,
- inform the user about open preconditions of activities (e.g. missing input documents),
• suggest solutions to conflicts between activities,
• reason about time aspects of the workflow,
• explain the planning process and justify the planner decisions.

The intelligent assistant has to do a lot of reasoning about semantic aspects of workflows. This requires that it has access to a declarative description of the workflow models. We therefore suggest to use a propositional representation of workflow meta data and to represent workflow activities as plan operators. This is illustrated by the following example - a specification of the “approval”-activity mentioned in Figure 1:

```prolog
clerk(frank)            % causal model
uses_as_editor(frank,emacs)

ACTIVITY approval {     % primitive activity
  REQUIREMENTS
  document(Form)
  EFFECTS
  decision(yes) v decision(no)
  ACTION
  start_app(Editor,document(Form))
  TEMPORAL CONSTRAINTS
  document(Form) during approval
  approval meets decision(_)
  SEMANTICAL CONSTRAINTS
  clerk(Actor)
  uses_as_editor(Actor,Editor)
}
```

The first two lines define facts about the causal model. They tell the system that there is a clerk, called “Frank” and that Frank uses Emacs as his editor. We propose that all static information (e.g. the hierarchical structure of an organization) is specified this way. The remaining lines of the example define a primitive activity “approval”. It has a single precondition: An input document must be present. The activity also has a postcondition - a condition, that is known to hold after the activity has finished. In this case the postcondition is the disjunction decision(yes) v decision(no): Depending on the decision of the actor the activity can have two different outcomes. The next paragraph in the example defines the operational aspect of the activity. It specifies that an editor should be started with the input document. After that, some temporal constraints are given: The input document has to be available as long as the approval activity lasts and the decision is known to be available, as soon as the approval activity has ended. The last two lines of the example specify some semantical constraints restricting the possible values of the workflow activity’s parameters (in this case the agent performing the activity and the editor to be used). These constraints link the activity specification to the causal model defined at the beginning.

The above example shows how workflow activities can be represented as plan operators. This implies that a workflow model can be interpreted as a (partial) plan specification consisting of the workflow’s goals, a set of plan operators (the workflow activities) and a set of temporal and semantical constraints.

The plan specification language sketched so far is not meant to be directly used as a workflow modelling language but as an intermediate language. It should preferably have a declarative nature to directly support the (meta) reasoning processes needed for the planning and execution of workflows. It should also be powerful enough to “simulate” any other workflow language. Theoretically, any workflow
specification language could then be effectively transformed into this declarative formalism.

3.3. Challenges to the assistant planner

Unlike typical AI planners this planner is not supposed to control an agent that has little or no knowledge (e.g. a robot). Instead, it should serve as an interactive support tool for the human workflow developer. This implies the following requirements:

- **least-commitment planning** The planner must not unnecessarily restrict the workflow designer. The ordering of the activities should only be constrained in case of conflicts, as in a *nonlinear* planner (McAllester and Rosenblitt, 1991).

- **conditional planning** The planner must be able to deal with incomplete knowledge. Workflow activities can have multiple outcomes (e.g. depending on the adhoc decision of a human being). In order to handle this situation the planner must be able to do *conditional planning* (Peot and Smith, 1992).

- **hierarchically structured plans** Traditionally workflows are defined in a hierarchical manner (Jablonski and Bussler, 1996) (Fig. 1). This has to be taken into account for the plan representation.

- **reuse of plan fragments** Since the system uses hierarchically structured plans, it is reasonable to save workflow plans and plan fragments into a repository and to reuse them later on (e.g. as subworkflows in other workflows).

- **abstract plan templates** A workflow model has to abstract from certain parameters (e.g. agent, documents) in order to be reusable. This must be reflected in the plan specification language (e.g. by permitting variables).

- **sensor actions** It must be possible to define sensor actions which enable the system to gather information about the world (e.g. the effects of an activity or semantic failures).

- **reasoning about time** Time is a critical factor in business processes. Thus, a WFMS must be able to reason about the duration of activities/workflows and inform the user when a time limit is in danger. This can be supported by a *temporal planner* (Rutten and Hertzberg, 1992).

- **multiagent planning** In many cases business processes require the coordinated work of several agents. Hence, the assistant planner must be able to construct *multiagent plans*.

Since our planner uses a STRIPS-like representation of operators (Fikes and Nilsson, 1971) it can automatically identify the inter-task dependencies (causal links) of a plan (McAllester and Rosenblitt, 1991). These dependencies link together the activities that establish and use assertions about the state of the workflow execution. One such assertion could, e.g., be the presence of a certain document.

These assertions provide important meta data. In the scenario we described in Figure 7, the system has used dependencies for identifying the activity that should have created the missing document and by not doing so caused the semantic failure.

Our planner is supported by a reason maintenance system (RMS). An RMS is a special deductive knowledge base. In our case it stores the dependencies between workflow activities in the form of a so
called dependency network and reasons about them. The RMS also provides useful interface functions to the planner: it can, e.g., tell the planner about open preconditions, required ordering constraints, or whether a plan is complete.

We not only suggest a plan representation for the object level processes but also to model meta level processes as plans. These meta plans contain meta operations such as plan, replan, check_consistency, notify_agent, or the sensor actions mentioned earlier. With the proposed declarative representation of the meta level the system is able to do introspective meta reasoning. As in other AI meta level architectures [see (Maes and Nardi, 1988)] both levels of this architecture are linked via so called reflection mechanisms. This enables computations on the meta level to control the execution of object level processes.

4. Related Work

There have already been several attempts to extend WFMSs with failure handling features. Most of them try to adopt (extended) transaction models known from database management systems (Eder and Liebhart, 1995; Leymann, 1995; Worah and Sheth, 1997) One of these systems, WAMO, was briefly described when we discussed the case of failure handling.

The prototypical WFMS MOBILE (Jablonski, 1994) allows the user to define his own control flow structures by means of Petri net specifications for later use in his workflow models (Jablonski and Bussler, 1996). This can be seen as a first step towards a user definable workflow meta model. However, unlike our system, MOBILE’s meta model cannot be dynamically adjusted at runtime.

A completely formal specification of a workflow description language and its execution semantics (based on relational algebra) was presented in (Schwab, 1993). Contrary to ours this representation does not allow to do meta reasoning.

The PLOPP! system (Lindner, 1994) is an interactive assistance planning system which is able to support humans when designing process plans. PLOPP! also uses a least commitment planner supported by a reason maintenance system, PNMS (Lindner, 1991). But compared to our system PLOPP! can neither explicitly reason about time, nor can it do conditional planning. Both of these features are uses a temporal extension of an RMS, the so called ATMS (Beckstein and Geisler, 1994).

Another interactive planning tool, called CAPLAN, was described in (Weberskirch, 1995). CAPLAN also takes advantage of an RMS, but uses it for the maintenance of dependencies between planning decisions (i.e. it stores metadata about the planning process itself). Based on these dependencies the system can, e.g., do dependency-directed backtracking, when the user rejects a planner decision.

5. Conclusion

We have shown that existing WFMSs still lack some important features, esp. intelligent failure handling mechanisms. This is due to the static and only implicitly defined meta model of these systems. We have argued that intelligent failure handling needs semantically enriched workflow models. This is only achievable by means of an explicit, dynamic, and user accessible meta model. We, therefore, have proposed a meta level architecture for WFMSs providing an explicit and dynamic meta model. This complex architecture is supported by an intelligent assistant based on established AI techniques. The system uses a plan representation for workflow models. With this representation the system is not only able to do intelligent failure handling but can also support the workflow modelling and execution processes.

We are currently in the process of defining specification languages for workflows, the meta model, and for the causal model. The implementation of an appropriate assistant planner which can use this representation is under way. The logical next step will be the design of a convincing intelligent assistant...
around this planner. One of the still open questions is how domain independent the assistant can be. We will also take care of an appropriate buildtime/runtime environment. Because of the enhanced complexity of the meta level architecture, an intuitive and easy-to-learn user interface is most important (visualization and monitoring tools, graphical workflow editor, reactivity). This should then result in a prototypical workflow system that convincingly demonstrates the practical advantages of the proposed meta level architecture for workflow management.

6. References


