Customizing Internal Activity Behaviour for Flexible Process Enforcement

Belinda M. Carter, Joe Y.-C. Lin, Maria E. Orlowska

School of Information Technology and Electrical Engineering
The University of Queensland
St Lucia, Brisbane 4072, Queensland

{bcarter, jlin, maria}@itee.uq.edu.au

Abstract

Workflow technology has met with success in a variety of industries, although several limitations have emerged. One such drawback is the inflexibility of specification languages, including a lack of support for inter-task dependencies. Expressiveness of the specification language is believed to be a determining factor of workflows applicability and its industrial value as solution for process support.

This paper attempts to address this limited language expressiveness by suggesting an alternative approach to modelling that more accurately captures behavioural information about tasks and enables greater precision when modelling inter-task dependencies.

Current workflow technology associates one generic, predefined finite state machine with each activity in a process, and inter-task dependencies of the type 'completion of one activity triggers scheduling of the next activity' are also enforced.

The potential improvement relaxes these constraints to enable the specification of user-defined finite state machines to represent each activity and support the modelling of inter-task constraints at the activity state level. In this paper, we present an introduction to this modelling extension and demonstrate the applicability of existing workflow verification algorithms to these more descriptive process models.

Keywords: Workflow, Business Process Modelling, Verification, Task Behaviour, Inter-task Dependencies.

1 Introduction

Workflow technology has been used in practice for managing process oriented business activities for many years. However, enterprises are undergoing rapid and significant changes, and current workflow technology is no longer sufficient to manage complex business processes and satisfy business' requirements.

It is important to consider improvements and extensions to the current solutions to overcome the limitations in order to make workflow technology more flexible and limitations of workflow technology have emerged (Alonso and Schek 1996).

These criticisms include:

- Limited support for inter-task dependencies
- Inflexibility in specification languages
- Difficulty in modifying process dynamically
- Difficulty in integration of heterogeneous workflow systems

A number of recent enhancements dealing with dynamic modification of workflows (e.g. Sadiq 2000), pockets of flexibility (e.g. Sadiq, S., Sadiq, W. and Orlowska 2001) and smarter verification during the process design phase (e.g. Sadiq and Orlowska 2000, Aalst and Hofstede 2000) have made contributions towards overcoming some of these limitations. However the potential exists for further improvement.

In this paper, we attempt to address the issues of limited flexibility in the specification and support for inter-task dependencies in workflow modelling languages and corresponding implementations. In particular, we will investigate the specification of internal activity behaviour and of 'fine-grained' inter-task dependencies defined between activity states in the process model. While the proposed extension can support more flexible execution, the impact on process complexity and correctness should not be ignored. We will demonstrate that the enhanced process model can also be verified with an established algorithm (Sadiq and Orlowska 2000) that was developed for traditional (activity-level) workflow process specification.

2 What is an Activity?

Since this paper focuses on the modelling of internal activity behaviour and inter-task dependencies at the activity state level, it would seem appropriate at this point to clarify the definition of an activity. There appears to be no commonly accepted definition in the literature, although most converge on "a piece of work that forms one logical step within a process" (Workflow Management Coalition 1995) or "some work to be done by a person, by a software system or by both of them" (Casati, Ceri, Pernici and Pozzi 1995). Some publications differentiate a task from an activity, for example, defining that a task represents "a concrete run-time work request to a particular person to perform a specific activity" (Leymann and Atlenhuber 1994).

We will use the terms 'task' and 'activity' interchangeably, and define tasks as logical units of work within a process that may be either manual or automated but performed by a single workflow participant.

2.1 Granularity

It is important to consider the issue of granularity in order to define an activity more precisely. In a workflow process model, the finest structure is an activity and the coarsest is the process as a whole. A business process could be modelled as one activity or multiple activities. However, an activity boundary is generally defined in respect to one of three factors:

- What (data object) – An activity is associated with a set of data objects that are processed or manipulated by the activity. For transactional purposes, data objects being processed in one task may be isolated from the access of other activities.
- Who (performer) – Each activity has an associated performer or participant possessing the relevant skills or job role.
- When (temporal) – Each activity may have a set of associated temporal constraints or properties, such as the minimum, average, or maximum durations to be expended for the activity. For example, an activity may be defined such that one hour is spent performing a particular task.

Clearly, the granularity of an activity determines the complexity of the model and subsequent execution of the workflow process, since the performer is required to interact with the Workflow Management System (WFMS) for each individual activity. The workflow modeller should specify the process in order to achieve the most efficient performance while enforcing as many constraints as possible.

2.2 Execution

Each activity in a process has an associated finite state machine (FSM) to represent the execution of an instance of the activity. The FSM consists of a set of visible states and a set of transitions between these states (Sadiq 2000). States in the machine represent the internal conditions that define the status of an activity instance at a particular point in time (Workflow Management Coalition 1999).

The generic FSM presented by the Workflow Management Coalition is suggested to have the states of *Scheduled*, *Active*, *Completed*, *Suspended*, and *Terminated* (Workflow Management Coalition 1995) and is presented in Figure 1. Current workflow products enforce that the FSM is predefined for all activities in a process, although different products may vary in their choice of FSM.

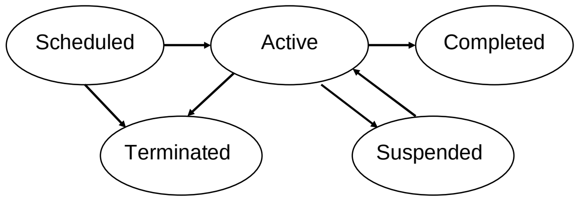

Figure 1: Example FSM for an Activity Instance

The WFMS is able to observe only the states represented in the FSM. Generally, the transitions between the states represent user events, and the workflow participant controls transitions from one internal state of an activity instance to another, for example, by indicating commencement or completion of the activity to the WFMS.

3 Control Flow

Current technology also enforces that the final state of an activity (generally ‘completed’) triggers the process flow to the next activity in the process, according the stored process definition. Transitions triggered from the completion of one or more activities are the only way in which the scheduling of tasks may be controlled.

In order to illustrate process execution, we introduce two types of objects: nodes and control flow. Each *node* is classified into three subclasses: task, coordinator and state. A *task*, graphically represented by a rectangle, represents the work to be done to achieve some objectives. It is also used implicitly to build sequence, fork, and synchronizer structures. A *coordinator*, graphically represented by a circle, is used to construct choice and merge structures. In addition, the *states* within an activity are graphically represented by an ellipse. *Control flow* links two nodes in the graph and is graphically represented by a directed edge. It represents execution order and flow between its tail and head nodes. The modelling objects are presented in Figure 2.

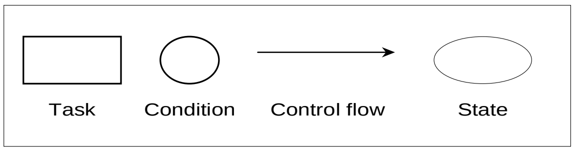

Figure 2: Process Modelling Objects

As previously mentioned, traditional workflow limits the control flow from the completed state of an activity to the scheduled state of the next activity. Figure 3(a) illustrates control flow in traditional workflow technology. Note that for simplicity, the scheduled, active and completed activity states have been abbreviated as ‘S’, ‘A’ and ‘C’, respectively, and the suspended and terminated states have been omitted. The dashed line indicates inter-task (or process level) control flow.

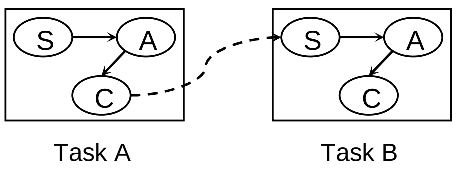

Figure 3(a): Traditional control flow

In relation to actual process execution, this restriction enforces that all activities ‘before’ the currently executing task(s) in terms of control flow must have been completed.

3.1 Limitations of Traditional Control Flow

The current approach is obviously restrictive in that only one type of inter-task dependency can be modelled and subsequently enforced.

Figure 3(b) illustrates an example of an inter-task dependency that cannot be modelled under the current workflow specification framework.

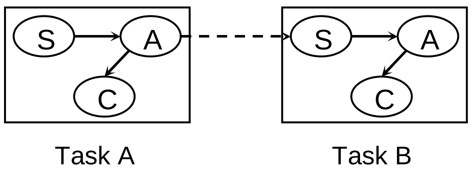

Figure 3(b): Relaxed Control Flow

We argue that there is a need for the ability to model such control flow to provide intermediate task behaviour management.

For example, consider a business scenario where an employee is able to email a document to the printing department for printing and binding and then pick up the document when it is ready. Assume that the printing department is busy and so cannot guarantee when a piece of work will be started, however it can guarantee that once a printer has started working on a particular document, the document will be ready for pickup in one hour. A business policy was thus created such that as soon as a printer commenced work on a particular document, an email was automatically generated and sent to the employee informing them that their document will be ready for collection in one hour.

This policy clearly requires the specification of an inter-task dependency such that the task to generate the email is triggered when the printing activity enters its 'active' state. However, the current modelling approach cannot enforce dependencies below the activity level. In order to model this scenario with the existing modelling technique, the granularity of the printing task would need to be reduced. This example is modelled under the current approach in Figure 4(a).

Clearly, this is not an acceptable way to model this scenario, as splitting the printing task into two tasks is neither intuitive in terms of modelling nor effectively enforceable in terms of execution, and the policy would probably not be enforced. Figure 4(b) illustrates the ideal modelling approach, in which the specification of trigger transitions at the task state level is permitted.

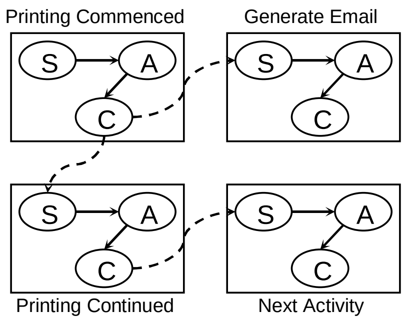

Figure 4(a): Business Policy Modelled Under Current Framework

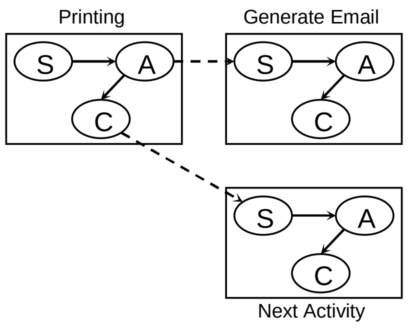

Figure 4(b): Business Policy Modelled Under Proposed Framework

4 Proposed Modelling Enhancement

Due to the absence of any constraints on activity definition, we are investigating an alternative 'fine-grained' approach to workflow process modelling that involves increasing the expressiveness of specification languages in order to more accurately capture task behaviour and model inter-task constraints. The proposed modelling enhancement is comprised of two aspects – the ability to more accurately model inter-task dependencies and the ability to specify customised FSMs to represent activity behaviour. Generally, tasks in a process are inter-related in such a way that initiation of one task is dependent on successful completion of a set of other tasks. However, we propose that under certain circumstances, advantages may be gained in terms of the overall process execution from modelling at the activity state level.

4.1 Flexible Inter-task Dependencies

Attie, Singh, Sheth and Rusinkiewicz (1993) describes inter-task dependencies as "the specification of constraints on the occurrence and temporal order of events, describing how the execution of a task is related to that of others (based on the state of execution of other tasks and their data outputs) and external events (e.g. messages from the user)." However, most process modelling languages are based on Petri Net like or equivalent semantics, where an activity is represented as one monolithic block that consumes all input information once it is started, and produces all outputs when it has finished. Inter-task dependencies therefore cannot be specified in terms of the execution states of constituent tasks; only 'complete → schedule' dependencies can be defined.

Modelling the workflow process at the task state level allows us to relax the restrictions on inter-task dependency modelling to enable the specification of inter-task dependencies on the basis of the internal states of constituent tasks, both in terms of individual dependencies and collections of dependencies (facilitated through the use of a coordinator). We refer to these relaxed dependencies as 'fine-grained' inter-task dependencies. Although the traditional specification of inter-task dependencies is sufficiently expressive to model most business scenarios, we argue that fine-grained dependencies have applications in the modelling of some business processes (for example, in the scenario presented above) and should be supported in modelling languages.

Although the specification of inter-task dependencies at the task state level seems to imply that dependencies can be defined between any two arbitrary states, not all dependencies may be practically enforced. We will classify the types of inter-task dependencies and reason them in the next section.

4.1.1 Classification of Inter-task Dependencies

Since inter task dependencies control the order of execution of tasks in a process, it would be appropriate to begin our classification of inter task dependencies with an appreciation of the possible ordering options under the current modelling approach. Ordering of tasks in current workflow models can comprise sequential, parallel, or conditional execution. The parallel execution is facilitated through the use of fork and synchroniser coordinators, and the conditional execution is facilitated through the use of choice and merge coordinators.

Each of these types of ordering is enforced through dependencies of the form 'Y cannot be scheduled until X completes', which we will term *typical trigger transitions*. This is the only class of inter-task constraint that is currently specifiable in most modelling languages. However, we observe that typical trigger transitions constitute just one class of possible inter-task dependencies. The proposed classification scheme of inter-task dependencies is illustrated in Figure 5.

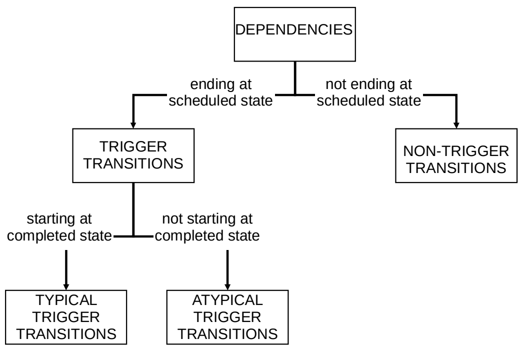
DEPENDENCIES
ending at scheduled state
not ending at scheduled state
TRIGGER TRANSITIONS
NON-TRIGGER TRANSITIONS
starting at completed state
not starting at completed state
TYPICAL TRIGGER TRANSITIONS
ATYPICAL TRIGGER TRANSITIONS

Figure 5: Classes of Inter-task Dependencies

We observe that two types of inter task dependencies exist – one in which the transitions trigger the scheduling of some activity, and the other where they do not. We label inter-task dependencies where the transition ends at a scheduled state as *trigger transitions*. As already noted, *typical trigger transitions* describe the special case where the transition begins at a completed state and ends at a scheduled state.

Atypical trigger transitions are not intuitively specifiable with the current modelling approach, and can only be enforced through reducing the granularity of the tasks involved in the dependency, which is undesirable for reasons already discussed and does not really capture the semantics of the constraint. Note that it does not make sense to specify an atypical trigger transition from a scheduled state to a scheduled state, because the transition would fire immediately (assuming the scheduling of a task requires a negligible amount of time); this constraint would be more intuitively modelled through a fork coordinator. Therefore, we anticipate that atypical trigger transitions will effectively always be defined from an active state to a scheduled state.

The example presented in Figure 4(b) illustrates the use of an atypical inter-task trigger transition to model a business policy. Inter-task trigger transitions are also useful in any otherwise sequential execution scenario where there are no data or temporal conflicts and where knowledge that the first task has commenced is enough to satisfy business policy or legal requirements and schedule the next activity. In this way, all business rules and requirements can be satisfied while allowing maximum parallelism between activities in a process, ultimately expediting process execution.

Non-trigger dependencies represent a constraint between tasks that does not result in the scheduling of an activity. An example of a non-trigger dependency is that 'Task A cannot be completed until Task B is completed'. These dependencies are not able to be expressed under the current modelling framework and further investigation is required into the implications of these dependencies on process execution.

Many non-trigger dependencies may not be practically enforceable. For example, it may not be appropriate to specify a constraint on the completion of an activity performed by a human participant since it may result in abnormal interaction between the human and the WFMS. However, issues concerning interaction between the WFMS and workflow participants may be resolved at the implementation level. A comprehensive analysis of the types of non-trigger dependencies that are able to be practically enforced is a key area of future research.

4.2 Customisable Activity Behaviour

Just as activities may have different transactional, temporal, and resource properties, they may have different execution behaviour, and we believe that the generic FSM may not be the most appropriate way of representing certain classes of activities.

We are therefore investigating the relaxation of the restriction of one predefined FSM representing all activities in a process, and considering some of the implications of allowing a different FSM to be associated with each activity.

The proposal is to allow each activity to have a customisable FSM that accurately captures its execution behaviour as a property. For example, the execution behaviour of a task may be more accurately modelled through zero or more than one active states. As another example, the suspended state may not be applicable to some tasks, while others may require multiple types of suspended states.

A full analysis into the circumstances under which this enhancement may be beneficial is yet to be completed. The issue of structural and semantic constraints on the composition of the customisable FSMs will also require further investigation. However, we have identified preliminary syntactic correctness criteria for each task FSM, as detailed below as part of the verification process.

4.3 Structural Verification of the Proposed Modelling Enhancement

While the specification enhancement allows us to model with greater precision, the workflow process model could potentially become much more complex.

It is possible to create error situations when specifying complex business processes, even under the current modelling framework in which each activity must be completed before process execution can proceed past the activity and each activity is represented by a pre-defined FSM. Verification algorithms based on graph reduction techniques have been successfully applied to detect structural conflicts such as deadlock and lack of synchronisation in process models defined under the current framework. In this section, the applicability of such an algorithm (i.e. Sadiq and Orlowska 2000) in detecting structural conflicts in these more descriptive process models will be demonstrated. Such modelling conflicts may lead to undesirable execution of some or all possible workflow instances, and possibly prevent further execution of the process. It is essential to correct such problems at the design phase rather than after deploying the workflow application.

We partition the verification process for the enhanced workflow structures into three phases. In the first phase, basic syntax checking is performed to ensure that all task FSMs are well formed. In the second phase, a transformation process is applied to translate the state-level process model into a traditional process model in a manner that preserves semantics and neither generates nor removes structural conflicts. In the third phase, the workflow model is analysed using existing algorithms to identify inconsistencies in the model that could arise due to conflicting use of modelling structures (specifically due to a mismatch of split/join types).

4.3.1 Activity FSM Syntax

All activity FSMs must conform to the following syntactic correctness properties:

- It is a directed, connected graph
- The only possible state types in the FSM are Scheduled, Active, Completed, Suspended, and Terminated
- All states are reachable from the Scheduled state
- It contains exactly one Scheduled state, exactly one Completed state, and some number of Active, Suspended, and Terminated states
- Beginning at the Scheduled state, exactly one path is possible through the Active states to the Completed state
- It contains only well-formed cycles. That is, all cycles are made from a pair of transitions where one transition originates from one of the Active states and ends at one of the Suspended states, and the other transitions begins at that Suspended state and ends at that same Active state. No other transitions are permitted to or from the Suspended states, and no other cycles are permitted in the machine.

4.3.2 Transformation Rules

Once the validity of all FSMs has been established, the transformation rules can be applied. Note that these transformation rules preserve semantics and neither generate nor remove structural conflicts. The following transformation rules are applied to each activity in the process:

1. Immediately discard all Suspended states and all transitions to and from these states.

The Suspended state necessitates a special type of iteration that is controlled by the workflow participant and completely unrelated to the process instance data. These special semantics cannot be captured by existing split and join structures. However, the integrity of the Suspended state(s) and associated transitions has already been established at the state level, plus these states cannot, by definition, participate in inter-task dependencies. Therefore, these states and corresponding transitions cannot possibly contribute towards a structural conflict at the process level, and removal of these nodes and transitions cannot introduce or remove any split or join structures into the graph. Note that the remaining FSM must be a directed, connected, acyclic graph with all states reachable from the Scheduled state.

Consider now the semantics of a task FSM. By definition, the instance must be in exactly one state at any time. All branch cases in the FSM, where more than one outgoing transition exists from one state to more than one other state (where the other states are internal to this activity), thus have implicit choice semantics. The next steps in the transformation process explicitly model these choice semantics inside the FSM, through the correct placement of the choice coordinators and merge coordinators.

2. For any states that possess two or more outgoing transitions to other states that are internal to this activity, a choice coordinator is constructed such that there is a transition from the state to the coordinator, and all original outgoing transitions of the state become outgoing transitions of the coordinator.
3. For any states that possess two or more incoming transitions from other states that are internal to this activity, a merge coordinator is constructed such that there is a transition from the coordinator to the state, and all original incoming transitions of the state become incoming transitions of the merge node.

Current models do not emphasise the internal workings of tasks (Sadiq and Orlowska 2000), and recall that modifying task granularity is a current mechanism for the enforcement of certain constraints under the current modelling framework. We therefore argue that every remaining state in the FSM is more or less equivalent in semantics to tasks specified in a current process model, and can be considered equivalent for the purposes of verification.

4. Transform every remaining state into a regular task.

This transformation replaces each original task in the process model with a well-formed workflow, free from the structural and grammatical errors presented in Sadiq and Orlowska (2000). (Note that structural conflicts can only arise when combining types of split/join structures, and only choice/merge structures have been introduced at this stage.) In the absence of any non-standard inter-task dependencies (i.e. atypical trigger transitions or non-trigger transitions), these segments of the overall models could be aggregated into blocks in current process models.

Inter-task dependencies have fork/synchronise semantics. That is, all inter-task transitions ending at a state must be fired in order to enter that state, and all inter-transitions originating at a state must be fired upon leaving that state. Combining the concepts of internal and external transitions yields that *all* inter-task (external) transitions leading to the state must be fired, along with *one* (if any) internal transitions that lead to the state, in order to activate a state. The corresponding situation applies for all transitions leaving a state.

The resulting graph is therefore correct in terms of implicit fork/synchroniser semantics. Note that the semantics of inter-task dependencies may be explicitly modelled through the correct placement of fork and synchroniser coordinators although this is not required for the purposes of verification (Sadiq and Orlowska 2000).

4.3.3 Application of Existing Algorithm

The placement of the split/join structures in the graph will then be analysed through the application of the existing verification algorithm (Sadiq and Orlowska 2000), which forms the third phase of the overall verification process.

Presented below are two examples of the transformation process being applied to a process model defined at the activity state level in order to detect structural errors.

For simplicity, the scheduled, active and completed activity states have been abbreviated 'S', 'A' and 'C', respectively, and the suspended and terminated states have been omitted.

The transformation illustrations remove the activity boundaries and create new activities that are each assigned a unique identifier. For example, the scheduled state of task 'A' is transformed into an activity with the identifier of 'A-Scheduled' which is abbreviated as 'AS'.

The transformation illustrated in Figure 6 demonstrates a process model on which an existing algorithm may be applied in order to detect a deadlock structural error.

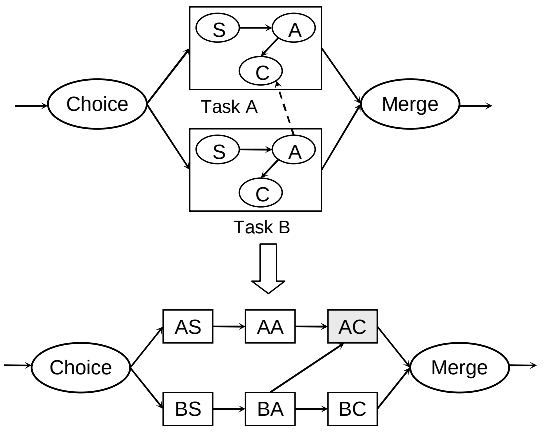

Figure 6: Deadlock Structural Error

The example shows that a deadlock occurs whenever there is an inter-task dependency between two tasks that are on mutually exclusive execution paths.

Figure 7 presents a transformation resulting in a process model on which an existing algorithm may be applied in order to detect a lack of synchronisation structural error.

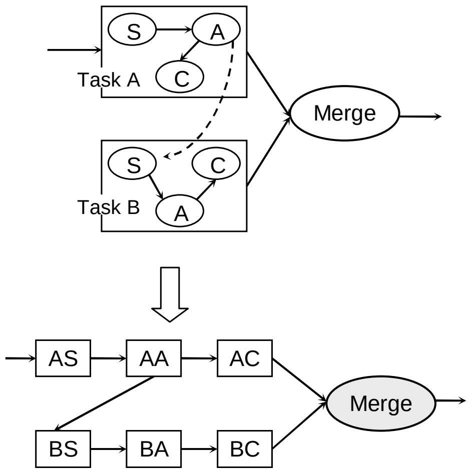

Figure 7: Lack of Synchronization Error

This example illustrates that lack of synchronisation occurs when an activity creates another execution path via an atypical trigger transition and the parallel execution paths are subsequently merged.

4.4 Additional Considerations

We have already demonstrated that structural conflicts in process models defined at the state level may be detected through existing verification algorithms after a transformation process has been applied.

However, it is possible to define an inter-task dependency between two activities that is structurally sound but is impossible to execute. The issues that could impact on the specification and verification of these more descriptive process models can be broadly classified into three classes – FSM semantics, data flow, temporal constraints, and failure recovery.

4.4.1 Activity FSM Semantics

As already noted, the goal of a task FSM is to represent the behaviour of a task as defined through interaction between the user and the WFMS. As such, it is imperative to realise that task FSMs are complex structures with rich semantics that that must be carefully considered in order to prevent the violation of any semantic constraints in the enhanced (state-level) process models.

Clearly, one example of a semantic constraint is that each activity must be scheduled. That is, it does not make sense to allow an inter-task dependency to 'trigger' or 'allow' any state in a task FSM if there is not such a dependency defined on the scheduled state of the task. Note that this requirement would be checked during the final stage of the verification process as there must be exactly one initial task (Sadiq and Orlowska 2000).

Moreover, no dependencies should be specified that prohibit normal user interaction with the system. Further investigation is required in order to detail the semantics and types of fine-grained inter-task dependencies that can and cannot be enforced.

4.4.2 Data Flow

Data flow is a semantic issue that needs to be considered when enabling inter-task dependencies to be defined at the task state level. Data is potentially produced and/or required in each of the active states in an activity. It should thus be possible to define the exact data item required and produced by each active state, as opposed to the task as a whole. Consider for example the process fragment illustrated in Figure 8. Task A produces data item X, task B required data item X, and task B is scheduled when task A has commenced. Even through the states where X is produced and required can be reached along a directed path, there is obviously potential for data conflict, namely in the case where B becomes active before A is completed.

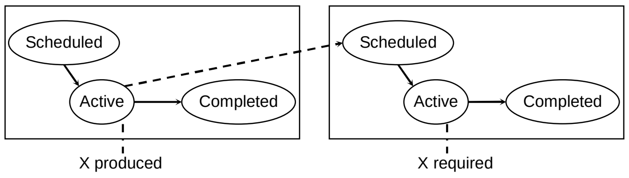

Figure 8: Data Conflict Produced by an Atypical Trigger Transition

4.4.3 Temporal Constraints

It should also be noted that the introduction of fine-grained inter-task dependencies may have implications in terms of temporal constraints.

For example, it is obviously not useful to define a constraint between tasks A and B such that B cannot complete until A completes if there is also a temporal constraint in place that B must or will always require a shorter time than A to complete. This situation is illustrated in Figure 9.

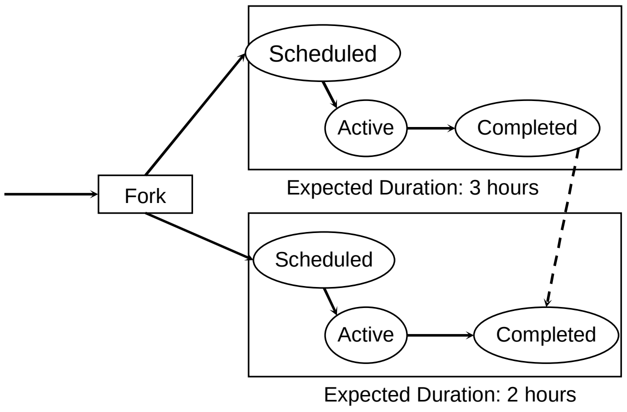

Figure 9: Potential Temporal Conflict at State Level

4.4.4 Failure Recovery

Another issue that should be considered when defining atypical trigger transitions is the possibility of failure (i.e. occurrence of system or application errors). Because the introduction of atypical trigger transitions into the process model violates the rules of (all-or-nothing) atomic execution, it is possible that execution may have been able to proceed past the task in the process where the problem was experienced.

In the example illustrated in Figure 4(b), the notification was generated as the printing activity commenced. If there had been a problem associated with the printing activity, the notification would have already been generated. This situation would certainly constitute a failure in terms of the business logic.

Failure recovery under the enhanced modelling framework is therefore a semantic issue that must be considered on a case-by-case basis and may not be able to be automated inside a verification algorithm. Failure to consider this issue may result in a complex situation requiring a cascading semantic abort that may need to be handled externally.

5 Conclusion

In this paper, we attempted to address the issue of limited flexibility in and support for inter-task dependencies in process specification languages by presenting an alternative approach to modelling that more accurately captures behavioural information about tasks and enables greater precision when modelling inter-task dependencies.

In particular, we proposed that advantages could be gained by allowing 'fine-grained' inter-task dependencies to be defined at the task state level and a customisable FSM to be associated with each activity.

However, we noted that the advantages of the fine-grained modelling were not forthcoming without an equivalent increase in the complexity of the process model and in verifying the enhanced process models. We demonstrated the applicability of an existing verification algorithm in ascertaining the structural integrity of the more descriptive process models. We then outlined a number of issues that should be considered in order to complete the verification of the complex process models and make a more accurate assessment on the value of the proposed modelling technique.

The areas of future research that could be based on the work presented in this paper include:

- Formalisation of inter-task dependencies and the transformation process for subsequent verification,
- A full investigation into implication of the proposed modelling technique into the areas of data flow, temporal constraints, and failure recovery, and
- A full analysis of the types of inter-task dependencies that can and cannot be enforced in order to preserve normal interaction between the user and the WFMS.

6 References

Aalst, W.M.P. van der and Hofstede, A.H.M. ter. (2002): An Alternative Way to Analyze Workflow Graphs. In *Proceedings of 14th International Conference*, on *Advanced Information Systems Engineering* (CAiSE'02). Proceedings Lecture Notes, 2002, pp. 757-60.

Alonso, G. and Schek, H.-J. (1996): Research Issues in Large Workflow Management Systems. *Proc. of NFS Workshop on Workflow and Process Automation in Information Systems State-of-the-Art and Future Directions,* Edited-by A. Sheth, Athens, Georgia, May 1996.

Attie, P.C., Singh, M.P., Sheth, A. and Rusinkiewicz, M. (1993): Specifying and Enforcing Intertask Dependencies. In *Proceedings of the 19th VLDB*, Dublin, Ireland, 1993.

Casati, F., Ceri, S., Pernici, B. and Pozzi, G. (1995): Conceptual Modeling of Workflows. In *Proceedings of 14th Object Oriented and Entity-Relationship Approach international Conference*, Gold Coast, Australia, Springer Verlag Lecture Notes in Computer Science 341-354, 1995.

Leymann, F. and Atlenhuber, W., (1994): Managing Business Processes as an Information Resource. *IBM Systems Journal*, Vol 33, No2, 1994Sadiq, S. (2000): Handling Dynamic Schema Change in Process Models. In *Proceedings of the 11th Australian Database Conference*, Canberra, Australia. Jan 27 - Feb 02, 2000. IEEE Computer Society, 2000.

Sadiq, S., Sadiq and W., Orlowska, M. (2001): Pockets of Flexibility in Workflow Specifications. In *Proceedings of the 20th International Conference on Conceptual Modelling (ER2001).* Yokohama, Japan. November 27-30, 2001.

Sadiq, W. and Orlowska, M. (2000): Analysing Process Models using Graph Reduction Techniques. In *Information Systems*, Vol. 25, No. 2, pp. 117-134, 2000. Elsevier Science. June 2000.

Workflow Management Coalition (1995): *The Workflow Reference Model*, Document No. TC00-1003.

Workflow Management Coalition (1999): *Workflow Management Coalition - Terminology*, Document No. TC00-1011. Issue 3.0.