The use of STIMEVIS in Business Process Reengineering with Workflow Specifications

Nikitas A. Assimakopoulos
Department of Informatics, University of Piraeus,
80, Karaoli & Dimitriou Str., GR-185 34 Piraeus, Greece.
Fax: +301-4179064, E-mail: assinik@unipi.gr

Abstract
This paper presents a systemic approach methodology called STIMEVIS for Structured Viable Systems and Metasystems towards the specification, verification, and distributed execution of workflows based on state and activity charts. The formal foundation of state and activity charts is exploited at three levels. At the specification level, the formalism enforces precise descriptions of business processes while also allowing subsequent refinements. In addition, precise specifications based on other methods can be automatically converted into state and activity charts. At the level of verification, state charts are amenable to the efficient method of model checking, in order to verify particularly critical workflow properties. Finally, at the execution level, a state chart specification forms the basis for the automatic generation of modules that can be directly executed in a distributed manner.
Keywords: Structured viable systems, Metasystems, Workflows, Business processes

1 Introduction

"*Business Process (Re) Engineering*" (BPR) is an important driving force for *workflow management.* It aims at increasing the efficiency of business processes and making them easily and quickly adjustable to the ever changing needs of customers (Assimakopoulos, 1999b). Today, business processes are mostly modeled in a high-level and informal way. BPR tools such as the ARIS-Toolset serve to construct graphical representations of business processes, including organizational properties of an enterprise. Some BPR tools support the analysis of quantative metrics of business processes, such as turnaround time, if the required statistical data is provided.

In contrast to the specifications of business processes, *workflow specifications* serve as a basis for the largely automated execution of processes. Workflow specifications are often derived from business processes specifications by refining the business processes specifications into a more detailed and more concrete form (Assimakopoulos, 1988, 1999b). Automated and computer-assisted execution means that a *workflow management system* (WMS) (Hsu, 1995) controls the processing of work steps-denoted *activities*- which have to be performed in the workflow. Some activities may have a manual or intellectual part, to be performed by a human. But the workflow management system is in charge of determining the (partial) invocation order of these activities. In contrast to the business processes specifications, this requires a

formal specification of control flow and data flow. Based on organizational properties of an enterprise, *roles* are assigned to those activities that require human interaction. At runtime, a human *actor* who can fill the given role is chosen to execute an activity.

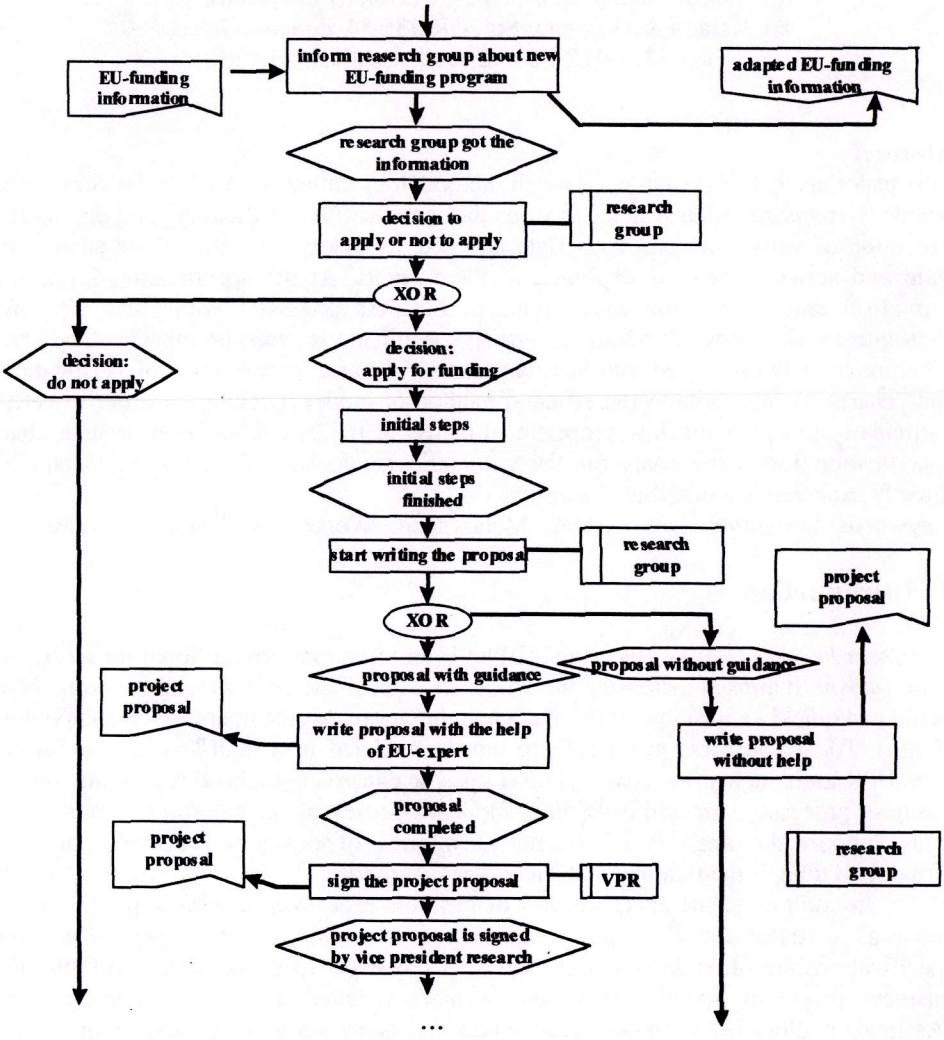

Fig. 1: ARIS specification of the Business Process 'EU project proposal'

Figures 1 and 2 show a part of a business process specification for a university research group submitting R&D project proposals to the European Union (EU). The specification of Fig. 1 has been created with the BPR tool ARIS-Toolset, whereas Fig.2

shows the same part of the specification given as a state chart, i.e., by using a formal specification method.

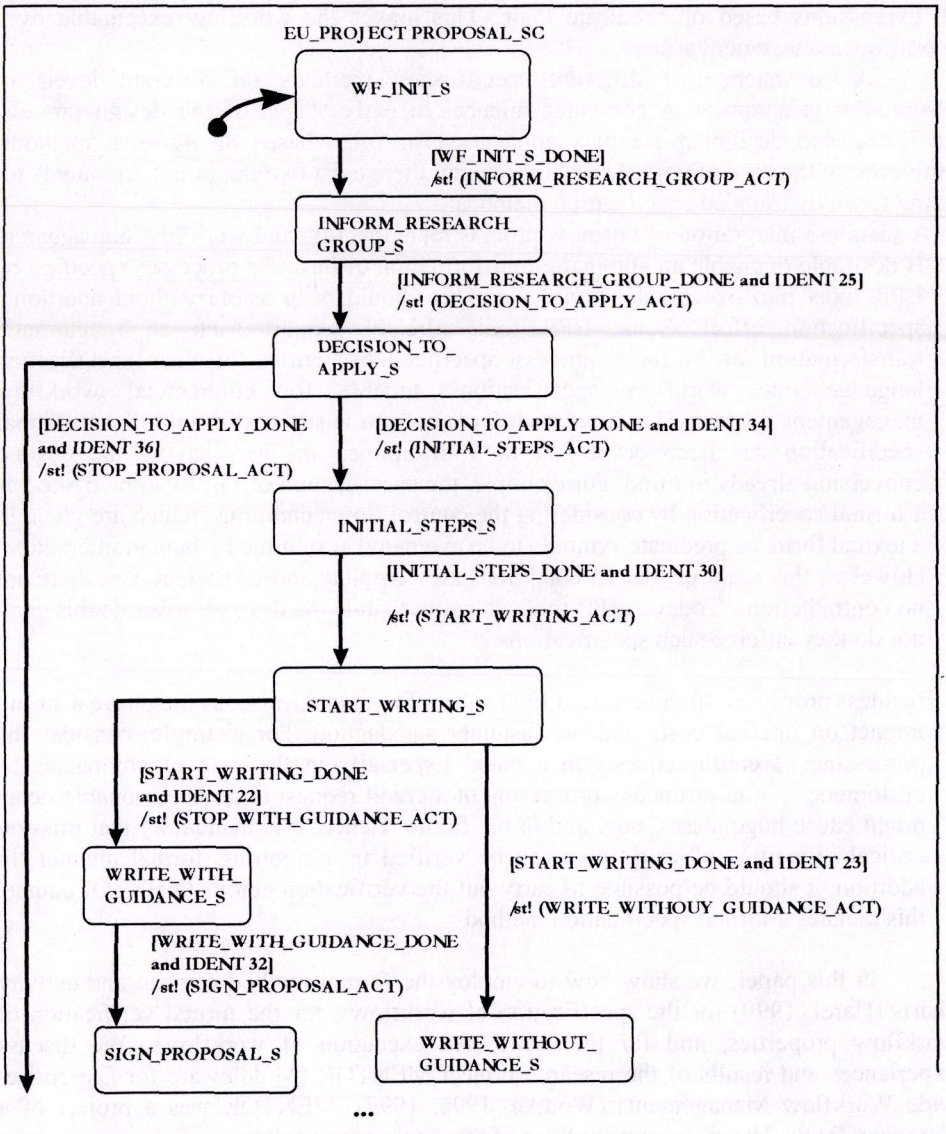

Fig. 2: State Chart Specification of the Workflow 'EU project proposal'

Both specifications show activities such as signing the proposal by the university's vice president of research, and both show the control flow in a graphical

way. In Fig.1, conditions for choosing different paths in the execution are given as plain text. Therefore, the specification might be incomplete and ambiguous. In Fig.2, the formal specification method of state charts requires all conditions to be explicitly given in expressions based on predicate logic. This makes the workflow executable by a workflow management system.

A coexistence of different specification methods on different levels of abstraction is common in computer science. In early stages of the design process, analyzing and designing complex applications is often based on informal methods. However, in the case of workflow management, there exist two important arguments for using formally founded specification methods:

- A seamless integration of business process reengineering and workflow management is desirable to enable an automatic transformation of business processes specified by BPR tools into executable workflows. This should be possible without additional specification effort. Some BPR tools already support such an automatic transformation of business process specifications written in their specification language into workflow specifications suitable for commercial workflow management systems. However, such a conversion is successful only if the original specification has been developed in a disciplined manner, having the desired conversion already in mind. For example, the specification of Fig.1 can be treated as a formal specification by considering the control flow conditions, which are given in a textual form, as predicate symbols to be evaluated at runtime by human interaction. However, this requires that all conditions are complete and consistent, i.e., there are no contradictions. Today's BPR tools do neither guide the designer towards this goal, nor do they enforce such specifications.

- Business processes often represent high values for an enterprise, as they have a strong impact on internal costs and on customer satisfaction. For example, consider the processing of credit requests in a bank. Especially in the case of companies as customers, an erroneous processing of a credit request or an unreasonable delay might cause huge losses, now and in the future. Hence, it is mandatory that mission critical properties of workflows can be verified in a rigorous, formal manner. In addition, it should be possible to carry out the verification automatically. Of course, this dictates a formal specification method.

In this paper, we show how to employ the formal method of state and activity charts (Harel, 1990) for the specification of workflows, for the formal verification of workflow properties, and for the distributed execution of workflows. We discuss experiences and results of the research project MENTOR (Middleware for Enterprise-wide Workflow Management) (Wodtke, 1996, 1997). MENTOR was a project of a European Bank. The main contributions of this paper are as follows:

- By using the properties of STIMEVIS multimethodology (Assimakopoulos, 1999a) and especially its designing tools (Panayotopoulos, 1987), we express the state and activity charts as our specification method, we are able to automatically verify

mission critical workflow properties with the method of model checking. Model checking is a well known approach in the field of reactive systems. Our contribution is to exploit model checking in the area of workflow management.

- Tools for formal specifications, including tools for model checking, typically have a centralized view on the specification. However, in practice huge and complex workflows have to be executed in a distributed environment. This is due to the decentralized structure of an enterprise or due to the different organizations that participate in a workflow spanning several enterprises. In order to integrate these different views and requirements, we have developed a method for partitioning a centralized workflow specification in an automated manner. Again, the formal nature of state and activity charts can be exploited for proving that the partitioned specification is equivalent to the original, centralized one. Therefore, properties proven for the centralized specification provably carry over to the partitioned one.

The paper is organized as follows. Section 2 summarizes basic requirements on workflow specification methods, and Section 3 gives a brief overview of specification methods used in research and industry. Section 4 introduces state and activity charts and shows how to model workflows in this formalism. Section 5 presents our approach of utilizing model checking for the formal verification of workflow properties, and experiences with this approach are discussed. Section 6 presents our method for partitioning workflow specifications and discusses the corresponding distributed execution. In general, we focus on aspects of control flow and data flow which are specific to workflow management. We do not discuss the organizational aspects of an enterprise and do not consider issues of the underlying information systems infrastructure, e.g., an enterprise-wide data model.

2 Requirements on Workflow Specification Methods

In general, a 'good' specification method should have the following properties:

1. The method must be able to specify the control flow and data flow of a workflow in an unambiguous and complete manner. This enables the execution of the specification by the workflow management system.
2. The method must have a well founded basis. In particular, a mathematically defined semantics is required. This in contrast to methods having their semantics defined by the code of corresponding workflow management system. The better the formal basis, the more formal results and tools typically exist (e.g., for an automatic verification of properties of a workflow).
3. In order to support an incremental specification process, the specification method must support the stepwise refinement of specifications. Likewise, the composition of existing specifications into larger, more complex ones must be possible within the formal framework.
4. It should be possible to visualize workflow specifications in a graphical manner, and to animate a workflow execution accordingly.

5. It should be possible to modify specifications at runtime. This is important for ad-hoc workflows, where the set of activities to be executed along with the data and control flow between them can be modified dynamically during execution.
6. The quality of a specification method typically depends on the application scenario as well as on personal preferences of the workflow designer. Hence, it should be possible to automatically convert specifications given in one method into other methods. This is also important in heterogeneous workflow environments, for the interoperability of different workflow management systems, and for building comprehensive process handbooks (Bernstein, 1995).

3 Formal Method for Workflow Specification

This section gives a brief overview of specification methods used in products and research prototypes of workflow management systems. We distinguish the following types of methods: script languages, net-based methods, logic-based methods, algebraic methods, and ECA rules.

Script languages: Workflow specifications based on *script languages* contain control flow and data flow constructs which are specifically tailored to workflow applications. Such script languages are popular in current WMS products. They provide a compact representation and are therefore easy to use. Experienced workflow designers even seem to prefer script languages over graphical interfaces, which are also offered by almost all WMS products. A drawback of most script languages is their lack of formal foundation. Their semantics is mostly 'defined' by the code of the script interpreter used.

Net-based Methods: When a graphical visualization of workflow specifications has top priority, *state transition nets* are a good choice. In state transition nets, activities are represented by nodes, and control flow is represented by edges. In fact, almost all WMS products provide means for graphical specifications similar to state transition nets. However, an important question is whether these net-based methods have a formally founded semantics. Unfortunately, this is not the case for most WMS products.

Considering only net-based methods with formal foundation, we have to restrict ourselves more or less to *state charts* and *Petri nets*. Variants of Petri nets, especially *predicate transition nets*, are used in a number of research prototypes as well as in several WMS products (Deiters, 1994). Some workflow management systems use variants of Petri nets for the internal representation of the workflow engine. State charts (Harel, 1990) have received little attention in workflow management, but they are well established in software engineering, especially for specifying reactive systems. For the MENTOR project (Wodtke, 1996; Weissenfels, 1996), we have chosen state charts as a formal foundation for workflow specification. In section 4, we will discuss state charts in detail.

Logic-based Methods: For a *logic-based* specification of workflows or for specifying dynamic aspects of information systems in general, *temporal logic* is a commonly used

method. Temporal logic has an excellent formal basis. A potential problem is the execution of specifications in temporal logic if the expressive power is to high. In addition, it is hard to visualize specifications in temporal logic, and it is often impossible to transform them into other specification methods. Simple forms of temporal logic, especially *computation tree logic* (CTL), have been used as a formal model for specifying execution dependencies in extended transaction models, and are also important for the verification of properties of workflow specifications (see Section 5).

Algebraic Methods: Process algebras (Milner, 1989) have been considered only in the theory community and are not widely known in the field of workflow management. As an exception, is the specification language LOTOS, which is based on a process algebra, is extended towards workflow management. The drawbacks of algebraic methods are similar to logic-based methods, but logic-based methods are superior to process algebras in terms of automatic execution and the potential for formal verification. In addition, specifications given in a process algebra are often not intuitive and hard to understand for practitioners.

ECA Rules: Event-Condition-Action-Rules, shortly termed *ECA rules*, are used in active database systems (Widom, 1995) and have been adopted by a number of projects in the workflow area (e.g., Geppert, 1995). ECA rules are used to specify the control flow between activities. Like for other methods that are not based on nets, the graphical visualization of sets of ECA rules is a non-trivial task. Large sets of ECA rules are hard to handle, and a step-wise refinement is not supported. In terms of their formal foundation, ECA rules are typically mapped to other specification methods, especially variants of Ptri nets or temporal logic.

Many relationships exist between the above categories for a formal analysis of such relationships). Hence, beyond the "pure" approaches that fit into the given categories, hybrid methods have also been proposed. Such hybrid methods are especially attractive when a graphic representation of non-net-based method is desired. Because of its designated role as an industry standard, the Workflow Process Definition Language (WPDL) of the Workflow Management Coalition is an especially notable hybrid method. The Workflow Management Coalition, an industry consortium, aims at a unified terminology and a standardization of interfaces between key components of workflow management systems. WDPL, currently available as a draft only, can on one hand be viewed as a textual representation of (highly simplified) state charts or state-transition nets. On the other hand one can view single statements as (again highly simplified) ECA rules. Finally, a complete WDPL specification could also be interpreted as a script.

4 Workflow Specification with State and Activity Charts

4.1 Introduction to State and Activity Charts

In this section, we will describe the specification of distributed workflows based on the formalism of state and activity charts (Harel, 1990). State and activity charts have originally been developed for reactive systems (e.g., embedded control systems in automobiles) and have been quite successful in this area. They comprise two dual views of a specification.

Activities reflect the functional decomposition of a system and denote the "active" components of a specification; they correspond directly to the activities of a workflow. An *activity chart* specifies the data flow between activities, in the form of a directed graph with data items as arc annotations. An example for an activity chart is given in the left part of Fig. 3. It represents a simplified workflow for the processing of credit requests of companies in a bank. A request for a credit involves, among other things, the checking of the company's current credit balance, the company's credit rating, and a risk evaluation.

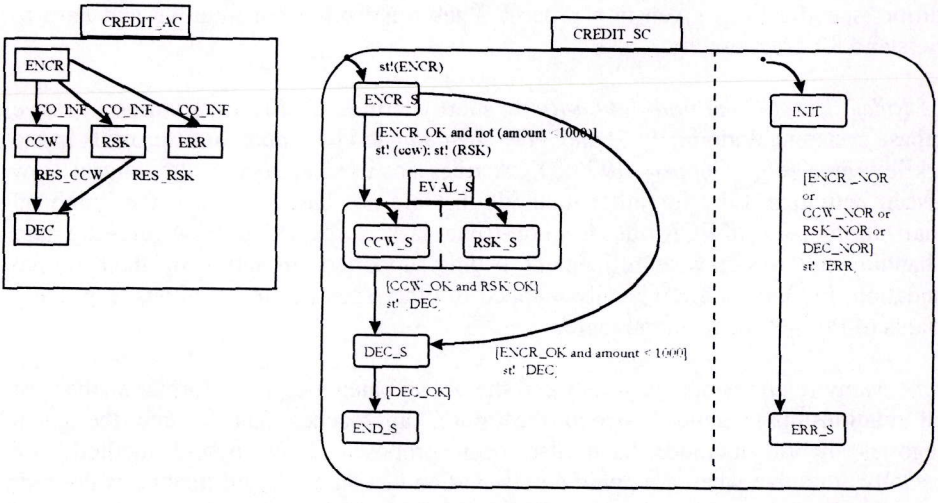

Fig. 3: Activity chart and state chart of the workflow 'credit processing'

The activity chart of Fig. 3 defines the activities *ENCR*, *CCW*, *RSK*, *DEC*, and *ERR*, and the data flow between these activities. Figures 2, 3 may also be designed using the drawing advantages of the Problem Structuring Methodology (PSM) (Panayotopoulos, 1987). In activity *ENCR* (enter credit request) the company's credit request is entered into a credit database. The company information *CO_INF* constitutes the data flow from *ENCR* to activities *CCW*, *RSK* and *ERR*. Activity *CCW* checks the company's credit worthiness by looking up the appropriate account and balance sheet data of the company. Activity *RSK* evaluates the potential risk that is associated with the requested credit. *RSK* takes into account the bank's overall involvement in the affected branch of

industry (e.g., the total of all credits already granted to computer companies) and the requested currency. Finally, the decision activity *DEC* records the decision that is eventually made on the credit request (i.e., approval or rejection); this would typically be an intellectual decision step based on the results *RES_CCW* and *RES_RSK* of the activities *CCW* and *RSK*. These four activities are complemented by a fifth activity *ERR*, which may be invoked as a generic error handler, for example, to stop the processing of the workflow when the credit applicant goes out of business.

State charts capture the behavior of a system by specifying the control flow between activities. A state chart is essentially a finite state machine with a distinguished initial state and transitions driven by Event-Condition-Action rules (ECA rules). Each transition arc between states is annotated with an ECA rule. A transition from state X to state Y, annotated with an ECA rule of the form E[C]/A fires if event E occurs and condition C holds. The effect is that state X is left, state Y is entered, and action A is executed. Conditions and actions are expressed in terms of variables, for example, those that are specified for the data flow in the corresponding activity chart. Conditions and events may also refer to states by means of special predicates such as $\mathrm{IN}(s)$ which holds if and only if state s is currently entered. In addition, an action A can explicitly start an activity, expressed by *st!(activity)*, and can generate an event E or set a condition C. Each of the three components of an E[C]/A triple may be empty. Every state change in a state chart execution is viewed as a single step; thus, state changes induce a discrete time dimension.

Two important additional features of state charts are *nested states* and *orthogonal components*. Nesting of states means that a state can itself contain an entire state chart. The semantics is that upon entering the higher-level state, the initial state of the embedded lower-level state chart is automatically entered, and upon leaving the higher-level state all embedded lower-level states are left. The capability for nesting states is especially useful for the refinement of specifications during the design process. Orthogonal components denote the parallel execution of two state charts that are embedded in the same higher-level state (where the entire state chart can be viewed as a single top-level state). Both components enter their initial states simultaneously, and the transitions in the two components proceed in parallel, subject to the preconditions for a *transition* to fire.

An example of a state chart is given in the right part of Fig.3 for the credit processing workflow. For each activity of the activity chart there exists a state with the same name, extended by the suffix *S*. *Furthermore, the workflow consists of three major sequential building blocks, ENCR_S, EVAL_S*, and *DEC_S*, where *EVAL_S* consists of two orthogonal components, *CCW_S* and *RSK_S*. Hence, the activities *CCW* and *RSK* are executed in parallel. For each activity, its outcome in terms of an error variable is available. For example, variable *ENCR_OK* is set to *true* after termination of activity *ENCR* if no error occurred. Otherwise, the variable *ENCR_NOK* is set to *true* and an error handling activity is started. Initialization and error handling of the workflow are specified in an orthogonal component.

4.2 Design Methodology for Workflow Specifications

From the MENTOR project, one may gain experience in using state and activity charts for specifying complex workflows in banking applications. From these studies, we have derived the following simple methodology for specifying workflows with state and activity charts. We consider two cases. In the first case, the initial workflow specification is developed as a state and activity chart. In the second case, a state and activity chart is derived from a specification of a business process, given in the specification language of a BPR tool.

4.2.1 Design with State and Activity Charts

When state and activity charts are chosen as the initial specification method, we proceed in the following steps:

1. *Definition of Activities:* First, the activities are defined. An activity might be a fully automated system call such as creating a new customer id, or a manual, interactive work step such as a memo by using an editor, or an intellectual work step such as deciding on a customer's discount.
2. *Assigning Activities to Roles:* For each activity, a corresponding role is assigned to it. This guarantees that at the time the workflow is executed, an adequate actor is selected for performing the activity.
3. *Definition of the Data Flow between Activities:* Here, we consider all data and all object identifiers which are relevant for the control flow. For example, in credit processing applications the amount of a credit request can determine subsequent processing steps.
4. *Coarse Definition of Control Flow:* In this step, the (partial) execution order of activities is determined. Activities with no precedence between them can be executed in parallel.
5. *Precise Definition of Control Flow:* For each activity, the preconditions for its execution have to be determined. Such conditions can refer to elements of the data flow (e.g., the credit amount), to the termination of other activities, or to temporal data such as deadlines.

The above steps can be iterated until a complete specification is determined. In our case studies, state and activity charts have proven their stability, as they enforce precise specifications on one hand, but allow a later refinement of the specification on the other hand.

4.2.2 Transformation of BPR Specifications into State and Activity Charts

When discussing the above methodology with business practitioners such as bank staff, we have encountered problems in accepting the formal nature of state and activity charts

for specifying business processes. Some practitioners prefer (informal) BPR tools because they are more intuitive to understand and easier to handle. Therefore, we have also investigated transformations from workflow specifications developed with BPR tools into state and activity charts. In this case, state and activity charts become a 'canonical', internal representation for workflows specified with various BPR tools. In addition, in such an environment state and activity charts can also serve as an interchange format for workflow specifications between workflow management systems that use different specification languages. This transformation approach is feasible only if the control flow conditions in the BPR specification (given as text) can be interpreted as predicate symbols of a formal model of conditions which are evaluated at runtime. Since typical BPR tools would also accept contradicting or incomplete conditions, manual interaction may be required for transforming the BPR specification into state and activity charts.

As an example, we have investigated the following three tier transformation. Initially, business processes have specified as event-process chains with the BPR tool ARIS-Toolset. In a second step, these specifications have been transformed into FDL, the specification language of IBM's workflow management system FlowMark. This step was performed by a tool contained in the ARIS-Toolset. In the third step, the FDL specification served as input for an automatic conversion into state and activity charts. Figures 1 and 2 show a specification developed with the ARIS-Toolset, designed again for better understanding the situation using the properties of STIMEVIS (Assimakopoulos, 1999b) and of course the corresponding state and activity chart that was automatically generated. The underlying business process was taken from a set of business processes which model the administrative processes. The example of Fig. 1 is (to a large extend) authentic. It describes the process of writing a project proposal addressed to the European Community (EU). The business process starts by gathering information about proposals for a new EU-funding program, performed by the administration. The gathered information is posted to the research groups. Based on this information, research groups decide whether a project proposal should be submitted or not. If the decision is positive, initial steps are taken, involving the administration. At the end of these initial steps, it is decided whether the project proposal should be written with help from EU-experts or by the research group alone. Subsequent steps are not shown.

According to the ARIS terminology, in Fig. 1 rectangles with rounded edges represent functions, the hexagons represent events. Parallelism in the control flow is expressed by *AND* connectors, *XOR* connectors specify alternative executions. Assignments of roles to functions are expressed by edges between the rectangles representing functions and rectangles representing roles, the latter having an additional vertical line. Input and output in terms of documents is visualized by the usual document symbol. Figure 2 shows the part of the generated state chart that corresponds to the ARIS specification of Fig. 1. Some information contained in the ARIS specification is mapped to an activity chart, which is not shown. In order to save space, the transitions given in the state chart have been partly simplified, e.g., *IDENT34* is actually defined as: *DECISION_APPLY_FOR_FUNDING.*

The completeness of the resulting state and activity chart, i.e., its direct executability, depends on the quality of the original specification of the business process. Hence, it is crucial that the specification developed with a BPR tool is as complete and precise as possible.

5 Utilizing Tools for Formal Verification

BPR tools often support the analysis of business from a business management point of view. In particular, they evaluate statistical data, e.g., about the execution times and the execution frequencies of activities. Verifying the correctness of workflow specifications, on the other hand, is still a rather unexplored issue of workflow management. The state of the art merely boils down to simulating workflow executions, based on a given set of input parameter patterns. The problem here is to find a set of input parameter patterns that is small enough to guarantee short simulation times but contains all necessary test patterns. To determine a good set of input parameters is up to the designer. A systematic approach to the verification of critical workflow properties must be based on formal verification methods. Such methods necessarily require a formal workflow specification as the Systemic Environment of Migrating Workflows (Assimakopoulos, 1999b).

5.1 Workflow Properties

Even with a formal approach, reasoning about a perfect match of workflow specifications and reality is impossible. Instead, we are aiming at the verification of selected, particularly critical properties of workflows which have to be represented formally as well. A number of proposals for the classification of properties of dynamic systems can be found in the literature. The probably most prominent one is Lamport's dichotonomy of *safety* properties ("some bad thing never happens") (Lamport, 1997). In our example of the project proposal workflow, two safety properties would be:

- A project proposal cannot be accepted and rejected at the same time.
- Funds cannot be transferred to the research group before a written cost statement is presented.

An example of a liveness property that considers possible terminations of the workflow would be:

- For each proposed project, either a contract is signed or the proposal is rejected.

Note that these properties refer to process executions and cannot be expressed as dynamic integrity constraints on the data of the underlying information systems. Hence, the required properties cannot be enforced by standard database mechanisms. In principle, active database systems may be able to address this problem, but they suffer from other drawbacks (see Section 3).

5.2 Specification of Workflow Properties

A prerequisite for the verification of workflow properties is the modeling of these properties in a formal language. For this purpose, variants of temporal logic are well established. Temporal logic extends predicate logic by temporal operators. With these operators, conditions about the past or future can be modelled, for example: 'If p was true in the past, q will be true at some point in the future.' A temporal logic with an expressive power suitable for properties of workflows (according to our experience) is *CTL (Computation Tree Logic)*. In *CTL*, each temporal operator (X,F,G,U) is directly preceded by an existential quantifier A. These quantifiers refer to the set of possible execution paths that branch out from the current state. The temporal operators have the following semantics: Xp ('neXt p') means that in the next step p holds; Fp ('Finally p') means that p will hold eventually; Gp ('p *U*ntil q') means that q will hold eventually, and until then p holds.

The informally stated properties of Section 5.1 for the EU project proposal workflow can be expressed in CTL as follows:

AG ¬ EF (REJECT_PROPOSAL_S & ACCEPT_ PROPOSAL_S)

and

AG (FUNDS_TRANSFER_S ⇒ COST_STATEMENT_S)

and

AG (((EF DECISION_TO_APPLY_DONE_S) &
(EX DECISION_APPLY_FOR_FUNDING))
⇒ EF (SIGN_CONTRACT_S | REJECT_PROPOSAL_S)).

The term AGp means that at all future time points of all possible executions p holds. EXp means that here exists at least one execution path where p holds in the next execution step on this path. EFp means that on at least one execution path p will eventually hold.

5.3 Model Checking

Once the critical workflow properties are formally stated, they can be verified against the formal specification of the workflow. Two different approaches are possible here. The first approach is using theorem provers, which automatically verify that a given specification has the desired properties. However, in many cases this approach is computationally infeasible. As a less powerful, but much more efficient approach, *model checking* can be used. In essence, model checking verifies whether a given finite state automaton (the workflow specification) is a model of a given temporal logic formula. The most efficient variant of model checking is known as symbolic model checking (McMillan, 1993). Symbolic model checking is based on a compact, symbolic representation of a finite state automaton in terms of an *ordered binary decision diagram (OBDD)*.

Because state charts are closely related to finite state automatons, model checking can be applied to state charts. Tools for symbolic model checking already exist (at least as research prototypes) and are used mainly in hardware design and for reactive embedded systems. In the MENTOR project it has been used a tool for symbolic model checking. When using this tool, main memory turned out to be the decisive bottleneck. Verifying the formulas given in Section 5.2 required approximately 60MByte of main memory (the *OBDD* consisted of 39.467 nodes), and determining the result (tautology in all cases) took between 1 and 11 seconds. Although the resource requirements were high, the example shows that symbolic model checking is suitable even for the interactive verification of workflow specifications of non-trivial size. However, verifying critical workflow properties by means of model checking requires the workflow designer to be familiar with CTL. As a consequence, we anticipate that at least for the near future, model checking will remain a tool for experts among the designers of workflows. But it is also reasonable to assume that especially for the design of complex, mission critical workflows such experts will in fact be consulted.

6 Distributed Workflow Execution

Formal verification methods have an inherently centralized view of a specification: independently of the formal specification method used, they are based on a specification which comprises the whole system and does not reflect aspects of distribution. Complex workflows will, however, often be executed in a distributed fashion for the following reasons:

- Applications with a high number of concurrently active workflows, e.g., in insurance companies or medical applications, will impose a high workload on the underlying workflow management system which is responsible for maintaining the state and context information of all active workflows. Scalability and availability demands thus that the workflow management system be distributed across several servers.
- In complex workflows, activities of different departments or even autonomous branches of an enterprise cannot be freely assigned to workflow servers. Instead, the execution of an activity is often restricted to the workflow server 'owned' by the department where the activity belongs to. Hence, the organizational decentralization of an enterprise may explicitly demand a partitioning and distributed execution of workflow.

We now present the method for the partitioning of workflow specifications. As we shall see, partitioning the state chart is the most challenging part. The partitioning transforms a state chart into a behaviorally equivalent state chart that is directly amenable to distributed execution in that each distributable portion of the state chart forms an orthogonal component. The first step towards a partitioned specification is thus called *orthogonalization.* The outcome of this step is another state chart that can easily be partitioned. The proof that this transformation is indeed feasible in that it

preserves the original state chart's semantics is cast into showing that the resulting orthogonalized state chart is a homomorphic image of the original specification. Note that we do not address the issue of 'streamlining' or optimizing business processes by altering the invocation order of activities. We are solely concerned with preparing workflow specifications for distributed executed.

6.1 Partitioning of State and Activity Charts

The partitioning of workflow specification consists of two elementary transformations: the partitioning of the activity chart and of the state chart. We assume that for each activity of the activity chart there exists an assignment to the corresponding execution role and thus an assignment to a department or business unit of the enterprise. Therefore, each activity can be assigned to a workflow server of the corresponding department or business unit. Consequently, the partitioning of the activity chart falls out in a natural way in that all activities with an identical assignment form a partition of the activity chart.

While this is straightforward, the partitioning of the state chart requires more than simply assigning states to partitions. State charts can be state trees of arbitrary depth where parent states may have assignments that differ from the assignments of their children states. Furthermore, transitions may interconnect states at different levels of the state tree and/or with different assignments. In order to partition a state chart without changing its behavior, we pursue an approach which is different from the partitioning of the activity chart and is organized in three major steps:

(1) *Assignment of states to activities*

In this step, each state is assigned to an activity of the activity chart. For each state, this assignment allows to derive to which execution role and department or business unit it belongs. In the case of nested state charts where a state can contain an entire state chart the higher level state will correspond to a higher-level activity chart with embedded subactivities.

(2) *Orthogonalization of the state chart*

For each state of the state chart an orthogonal component is generated. Each generated orthogonal component emulates the corresponding original state *S* by means of two states, *in_S* and *out_S*, which are interconnected by transitions in both directions. These transitions correspond to the transitions that lead to or originate from state *S*. The transition labels of these transitions are extended by conjunctive terms that refer to the source states of the original transitions. Figure 4 illustrates the orthogonalization for the state and activity chart of Fig. 3. For example, for the transition that interconnects states *ENCR_S* and *EVAL_S* the following transitions are generated: a transition from *in_ENCR_S* to *out_ENCR_S*, a transition from

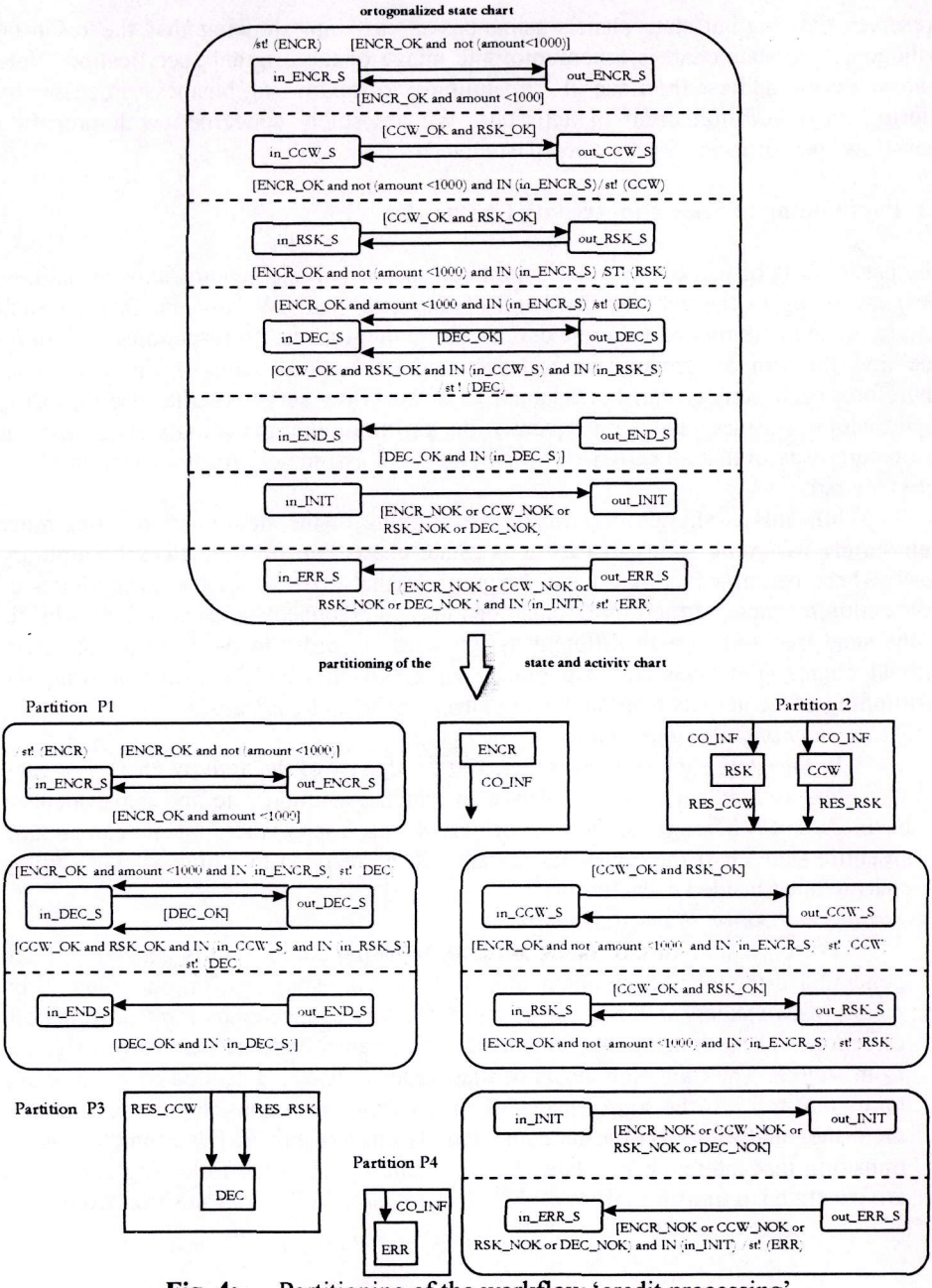

Fig. 4: Partitioning of the workflow 'credit processing'

out_CCW_S to *in_CCW_S*, and a transition from *out_RSK_S* to *in_RSK_S*. The first transition has the same condition component in its transition label as the original transition, *ENCR_OK and not (amount <1000)*. The other two transitions have the condition *ENCR_OK and not (amount <* extended by a conjunctive term *IN(in_ENCR_S)* as condition components and the start instructions for the corresponding activities as action components of their transition labels. This extension of condition components guarantees that in the orthogonalized state chart, all transitions which are generated from the same original transition fire at the same time. Consequently, in the example the activities *CCW* and *RSK* are started if and only if state *ENCR_S* is left, and *amount* is not less than 1000. If *amount* is less than 1000, the decision will be made without requiring an analysis of the company's credit worthiness and the risk, and state *DEC_S* is entered when *ENCR_S* is left. In this case, activities *CCW* and *RSK* are not invoked. Note that the orthogonal components in Fig. 4 have been automatically generated by our partitioning algorithm. This is the reason for the two transitions from state *in_ENCR_S* to state *out_ENCR_S* which could be merged into a single one with just *ENCR_OK* as condition component.

(3) *Assignment of partitions to workflow servers*

In this step, the orthogonal components that have been generated in step (2) are grouped into state chart partitions. The criterion for grouping a set of orthogonal components to a form a state chart partition is the assignment of their underlying original states to the same department or business unit of the enterprise.

Note that for the example of Fig. 4, we assume that the activities *CCW* and *RSK* belong to the state partition, whereas the remaining activities belong to different partitions. The data flow between activities belonging to different partitions is indicated by data flow connectors that start or end at the borders of the boxes that represent partitions. If a partition contains several orthogonal components which have formed a coherent part in the original state chart, it is possible to merge these components. Consider, for example, partition *P4* in Fig. 4. State *in_INIT* and *out_ERR_S* are left if one of the activities *ENCR, CCW, RSK* and *DEC* fails, i.e., the corresponding '*_NOK*' variable becomes *true*. The transition from *out_ERR_S* to *in_ERR_S* contains the term *IN(in_INIT)*. If we change the source of this transition to the state *in_INIT*, and remove the now unnecessary states *out_INIT* and *out_ERR_S*, we have reconstructed the corresponding part of the original state chart, namely the orthogonal component on the right side of the state chart in Fig. 3. For the sake of simplicity, we do not consider such merges in the rest of the paper.

6.2 Correctness of the Partitioning

The feasibility of the described transformation depends on the assumption that the semantics of the original state chart is preserved. What is needed is a formal correctness proof that this is indeed the case. We will concentrate on the correctness proof of the

orthogonalization (step(2)), which is the most critical step with regard to possible changes of the behavior of the state chart.

First, we will refer to a formally defined operational semantics of state charts, which is a simplified version of the semantics. It is tailored to our context of workflow specification. To this end, we define the set *SC* of *system configurations*. At each point, the system configuration describes the set of currently entered states and context, i.e., the current values of conditions, events, and variables that are part of the state chart. For the execution of a state chart, we define a step operator, *step*, which maps a system configuration to its successor system configuration.

Figure 5 illustrates the independence between the operational semantics of the original state chart and the operational semantics of its orthogonalized representation. We view both state charts as algebras with identical signatures. The carrier sets are the sets of system configurations, and the only operator is the step function. In order to distinguish between both carrier sets and operators, the carrier set and the operator of the algebra of the orthogonalized state chart (i.e., algebra 2) are denoted by *SC'* and *step'*, respectively. The interdependence between the two algebras is described by the mapping h_{SC}, which maps each system configuration of algebra 1 onto a system configuration of algebra 2. For the contexts this mapping is the identity mapping. With regard to the state configuration, h_{SC} maps each state *Z* that is currently entered onto a corresponding state *in_Z* of the orthogonalized state chart and each state *Z* that is currently not entered onto the corresponding state *out_Z* of the orthogonalized state chart.

If the diagram given in Fig. 5 commutes then algebra 2 of the orthogonalized state chart is homomorphic image of algebra 1 of the original state chart. In this case, we say that the orthogonalized state chart is *behaviorally equivalent* to the original state chart, i.e., has the same properties as the original state chart. For example, the activities are started at the same time whenever both state charts are executed under identical external input. The following theorem states the above formally:

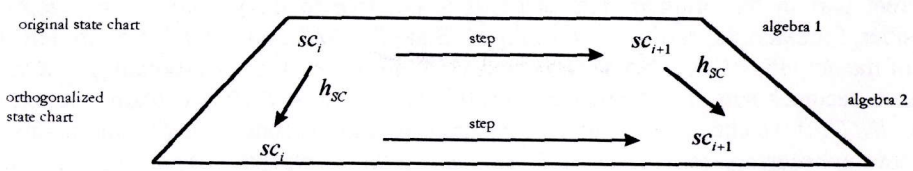

Fig. 5: Interrelationship between original state chart and orthogonalized state chart

Theorem 1

The mapping h_{SC}, which maps each system configuration of an arbitrary state chart *S* onto system configurations of the orthogonalized state chart *S'* is a homomorphism. That is, for each timestep $i, i \geq 0$ and for each system configuration $_{SC_i}$ of *S* the following holds:

$$h_{SC}(step(_{SC_i})) = step'(h_{SC}(_{SC_i}))$$

with *step* being the step operator of S, *step'* being the step operator of S', and both step operators being defined by the operational semantics of state charts.

6.3 Synchronization

In distributed workflows, the execution of workflow partitions has to be synchronized. For a partition to execute the next step, the system configuration of other partitions, i.e., their state and context information, may be necessary. This is the case if information written in other partitions is used in a local condition or read by a local action. In the original state chart semantics, all updates to the system configuration are immediately and ubiquitously available for the evaluation of conditions and the execution of actions. This scheme can be implemented in a distributed environment by broadcasting all changes of local system configurations to all other partitions after a step performed. The next step must not be performed before the corresponding information, the *synchronization data* from all other partitions, has been received.

In the following, we will discuss a simple communication scheme implementing the exchange of system configurations by sending synchronization messages between partitions according to the above rules. Issues of fault tolerance are considered in the next subsection. A more advanced scheme aiming to reduce the number and size of the required synchronization messages can be found.

The computation and exchange of synchronization data between partitions is performed by the following three steps. The first two steps take place in the workflow engine executing the sending partition, the third takes place in the workflow engine executing the receiving partition.

(1) **Collecting updates:** All updates to state and context information are collected. This involves the detection of local updates in the executing state charts and activities, and the temporary storage of update information.

(2) **Communicating update information:** The synchronization data is send to the receiving partitions. If no synchronization data exists, an empty message is sent.

(3) **Propagating the updates in the receiving partition:** The received synchronization data is used to update the system configuration of the receiving partitions. After synchronization messages from all remote partitions involved in the workflow have been received, a receiving partition is ready to perform the next step.

6.4 Fault Tolerance

We consider two kinds of failures, namely loss of messages and site failures. Consider the credit processing workflow and assume that the synchronization message that propagates the successful completion of the activity *ENCR* (i.e., the update that sets condition *ENCR_OK* to true) is lost. In this case, the activities *CCW* and *RSK* will never be started. A similar problem arises if only a subset of these activities receives the

synchronization message and thus starts executing. Both situations imply that the activity *DEC* will never be executed and the company will never be notified about whether the credit request is granted or rejected.

Now consider a different failure scenario in which the synchronization messages are successfully delivered but the site where the activity *ENCR* is executed fails immediately after sending the messages. As an effect of this site failure, the data that was inserted by the *ENCR* activity into credit database may become corrupted (unless sufficient steps are already taken to make the data recoverable). Then, the activities *CCW* and *RSK* would start executing, but would possibly not be able to access their input data in the credit database. All these unacceptable situations can be avoided if there exists a mechanism that detects that a message is lost or data is corrupted and retransfers the message or reconstructs the data, respectively.

One prerequisite for reliable communication is that all messages are saved onto durable storage, i.e., a reliable message queue, so that they can resent if necessary, even in the presence of site failures. However, this alone does not prevent the second one of the above two failure scenarios. To overcome this problem, it is necessary to combine the database update of the *ENCR* activity and the two synchronization messages into a single atomic unit. This property is exactly what distributed transactions can provide (Gray, 1993; Bernstein, 1997). Hence, in a workflow environment, transactions do not only refer to database updates but should also control message deliveries. A message should actually be delivered only upon the commit of the transaction. Thus; if the *ENCR* site fails before the commit of the transaction, the database update will be updone if necessary and the "pending" messages will be discarded. If the *ENCR* site fails after commit, the database updates will be redone if necessary and the messages that are marked as pending will be actually delivered to their recipients.

In case of a failure after commit, it is unfortunately impossible to identify, in retrospect, the exact point of failure. Therefore, it may happen that a message is both sent upon the original commit and resent again as part of the recovery. Such a generation of duplicate messages may be a problem as well, since each of the two message arrivals may trigger non-idempotent effects in the receiving activity. Unless the activity is explicitly implemented to cope with duplicate messages, a system mechanism is needed to detect and eliminate duplicates, for example, based on message sequence numbers.

In MENTOR, the above problems are addressed by employing a *TP* monitor, namely, Tuxedo (Primastava, 1994). A *TP* monitor essentially provides the following functionality:

(1) It offers facilities for saving messages onto durable storage. The *System Q* component of Tuxedo provides *enqueue dequeue* operations on *reliable message queues.* Messages that have been saved onto such a queue are resilient to site failures.
(2) It guarantees the atomicity of all operations within a transaction. To be precise, the atomicity guarantee refers to all operations that are invoked on transactional resource managers that can participate in a distributed transaction and comply with a

standarlized commit protocol like X/*Open XA*. This requirement is satisfied by most commercially elevant database systems and also by the System/Q component of Tuxedo, which thus serves as a special resource manager under the coordination of Tuxedo.

(3) By embedding the sender's enqueue and the recipients' dequeue operations in transactions, it can be insured that a message is delivered exactly once.

There is one more problem incurred by site failures that requires further recovery procedures beyond those provided by the *TP* monitor and the underlying resource managers. A site failure causes the loss of all information on the state chart execution of a failed workflow engine (i.e., the current state and the current context). This information is needed for continuing the workflow when the failed site is brought up again. To overcome this problem, all state transitions and updates to variables are recorded in a special *workflow log*. During the restart of a workflow engine, this log provides the necessary redo information for re-establishing the most recent state before the failure. Insertions into the workflow log are part of the Tuxedo-coordinated transactions which serve to propagate update information between the workflow engines. So, during recovery a failed workflow engine will be rolled forward exactly to its most recent synchronization point.

7 Conclusions

In this paper, we have advocated formal methods for the specification of workflows. We have demonstrated the usefulness of this approach by investigating the method of state and activity charts with the properties of STIMEVIS. For the workflow specifications, the advantages are twofold: First, formal methods enable the automatic verification of critical workflow properties. Secondly, a formally founded workflow specification is an important asset for the distributed (migrating workflows) execution of workflows: we have presented a transformation scheme for partitioning a centralized workflow specification such that the resulting partitions are again state and activity charts that are provably equivalent ti the original execution. In summary, we have achieve a seamless integration of the workflow specification, the use of tools for verifying workflow properties, and the distributed workflow execution.

In addition to the specification issues, we have investigated aspects of scalability, fault tolerance and availability. With the exception of a brief discussion of fault tolerance, these issues could not be discussed in this paper.

Further problems we want to consider in the future include the specification of transactional properties in workflows as well as dynamic modifications of workflows during execution:

- Many applications demand a facility for grouping a number of workflow activities into an atomic unit. Besides the required runtime mechanisms such as distributed transactions, compensation etc., the problem is to integrate the specification of transactions into the workflow specification and to analyze the impact of the

specification of transactions on properties of the original specification (Waechter, 1992).

In applications such as concurrent engineering or medical information systems, there is a strong need for dynamic modifications of processes and interoperability with *CSCW* (computer-supported cooperative work) tools whenever unforeseen situations arise and intellectual inetrception is required. Applications should allow to add or remove activities or even change the control flow between activities while a workflow is already in progress. In principle, the method of state and activity charts has been also used in the MENTOR project supports such dynamic modifications. However, the seamless transition from rigid workflow specifications to *CSCW* raises more demanding issues beyond the scope of this paper.

References

Alonso, G., Agrawal, D., El Abbadi, A., Kamath, M., Gunthor, R., Mohan, C. (1996). Advanced Transaction Models in Workflow Contexts, Proc. of 18th IEEE Intl. Conf. On Data Engineering, New Orleans, LA.

Assimakopoylos, N. 1988). The routing and cost of the information flow in a System, *Systems Practice*, 1(3), 297-303.

Assimakopoulos. N. (1992). A systems approach for hierarchically organized systems. *Analyse de Système*, 18(3-4), 3-13.

Assimakopoulos. N. (1999a). STIMEVIS : a systemic multi-methodology. *Human Systems Management*, to appear.

Assimakopoulos. N. (1999b). An ever-changing systemic environment for Migrating Workflows. Proc. of the 3rd Inter. Conf. on Computer Anticipatory Systems, Liège, Belgium, to appear.

Bernstein, P. A., Dellarocas, C., Malone, T. W., Quimby, J. (1995). Software Tools for a Process Hand book. In: (Hsu, 1995).

Bernstein, P. A. and Newcomer, E. (1997). *Principles of transaction Processing for the systems Professional.* Morgan Kaufmann.

Deiters, W., Gruhn, V. (1994). The Funsoft Net Approach to Software Process Management, International Journal of Software Engineering and Knowledge Engineering Vol. 4, No. 2.

Geppert, A., Kradofler, M., Tombros, D. (1995). Realization of Cooperative Agents using an Active Object-Oriented Database System, International Workshop on Rules in Database Systems, Athens.

Gray, J. and Reuter, A. (1993). Transaction Processing: Concepts and Techniques. *Morgan Kaufmann Publishers.*

Harel, D., et al. Statemate (1990). A Working Environment for the Development of Complex Reactive Systems, IEEE Transactions on Software Engineering Vol.16, No. 4.

Hsu, M. (1995). Special Issue on Workflow Systems. *Bulletin of the Technical Committee on Dta Engineering, IEEE,* 18(1).

Lamport, L. (1997). Proving the Correctness of Multiprocess Programs, IEEE Transactions on Software Engineering, Vol. 3, No. 2.

Liu, L., Meersman, R. (1996). The Building Blocks for Specifying Communication Behavior of Complex Objects: An Activity-Driven Approach, ACM Transactions on Database Systems, Vol. 21, No. 2.

McMillan, K. L. (1993). Symbolic model checking. Kluwer Academic Publishers.

Milner, R. (1989). Communication and Concurrency, Prentice Hall.

Panayotopoulos, A. & Assimakopoulos, N. (1987). Problem structuring in a hospital. *European J. of Operational Research*, 29: 135-143.

Primastava F. (1994). TUXEDO, An Open Approach to OLTP, Prentice Hall.

Widom, J., Ceri, S., Dayal, U. (1995). Active Database Systems, Morgan Kaufmann.

Wodtke D., Weissenfels J., Weikum, G., Kotz Dittrich, A. (1996). The Mentor Project : Steps Towards Enterprise-Wide Workflow Management. In Proc. 12th IEEE International Conference on Data Engineering, 556-565.

Wodtke D., Weissenfels J., Weikum, G., Kotz Dittrich, A., Muth, P. (1997). The MENTOR Workbench for Enterprise-wide Workflow Management, ACM SIGMOD Conference, Demo Program.

Waechter, H., Reuter, A. (1992). The Contract Model. In: Elmagarmid, A.K. (Editor), Database Transaction Models for Advanced Applications, Morgan Kaufmann.