UML Metamodel-based Workflow Modeling and Execution

Jens Brüning
University of Rostock
Department of Computer Science
D-18059 Rostock, Germany
jens.bruening@uni-rostock.de

Martin Gogolla
University of Bremen
Department of Computer Science
D-28334 Bremen, Germany
gogolla@informatik.uni-bremen.de

***Abstract*—In this paper, we present a UML metamodel-based approach for creating and executing workflow models. The workflow modeling language is introduced through its abstract syntax, and an evaluation shows how this language supports known workflow patterns. Some patterns can be expressed easier compared to established languages like EPCs or BPMN. Organizational and data aspects in workflow models can be described on the basis of the presented metamodel. The workflow models can be instantiated and executed with a tool realizing parts of the UML action semantics. At an early stage of design, our workflow models can be evaluated by testing scenarios with the used tool in combination with the developed workflow plugin. Employing the tool, dynamic aspects of the workflow process models together with data and organizational aspects can be evaluated. During execution of the workflow scenarios, the workflow models can be adaptively changed, and data can be captured and evaluated by formulating process mining queries with UML's OCL (Object Constraint Language).**

Keywords: Business process models, Business process metamodels, Workflow execution, Unified Modeling Language, Model Validation.

I. INTRODUCTION

Business process modeling gets more and more important with the increasing complexity and automation of business processes in companies and organizations. Business process models are used to document, restructure and optimize the processes. Furthermore, requirements for software and computer services that support the business processes are captured in these models. Nowadays flow oriented languages are frequently used for business process modeling like Event-driven Process Chains (EPC), UML activity diagrams and BPMN. These are languages based on Petri net token semantics which may restrict developers too much since they are following the principle "all executions paths are forbidden if they are not allowed in the model" [2]. Besides, the well-accepted workflow patterns are driven by Petri net token semantics. In contrast, declarative workflow models have a flexible background driven by design principles. That means, all execution paths are allowed if they are not explicitly forbidden. This declarative view is followed in this paper.

Compared to a Petri net-like modeling language, our work is an alternative possibility to express the workflow patterns in a declarative way with a metamodel based modeling approach using UML and OCL. In our view, this is a new direction in the context of workflow languages. There are several graph- or block oriented languages like UML activity diagrams [21] or BPEL [22] that are checked against the workflow patterns, but no language uses a declarative foundation to express them and to check declaratively formulated properties.

The literature provides many metamodels for business process modeling. Some of them are used for conceptual modeling to define elements of the workflow language and their interrelations [18, 14]. These metamodels can be further used to implement modeling tools [10]. The approach presented in this paper uses a UML metamodel along with the tool USE [9]. USE checks static properties of the workflow models during the modeling process by observation of OCL invariants. The modeler gets quick feedback as identified problems and the involved modeling elements are immediately indicated.

There are even more benefits to UML workflow metamodels with respect to dynamic properties. They provide means to define execution semantics. OCL invariants are used for system states and pre- and postconditions for operations. They describe the causal or temporal relationships between the modeling elements. During execution of the workflow model, the execution semantics is interpreted and disallowed flows of a process are forbidden. Furthermore, enabled activities can be identified and a worklist visualization is possible.

In our approach, a workflow plugin is implemented for the USE tool that presents the activities and the corresponding execution states in an appropriate way. The user can interact with the workflow plugin. Related data objects are presented to the user so that scenarios can be played through by the user interacting with the tool. Thus, the integrated workflow and data models can be tested before system implementation and the workflow models can be validated at an early design state.

It is also possible to test and compare different organizational models and configurations. Such comparison can be achieved by instantiating multiple workflows which are then tested in varying contexts.

Resource management is another important aspect. In our approach resources are allocated during the workflow execution. The shortness of particular resources may be identified in the system animation.

Our workflow models are flexible in two ways. On the one hand they are flexible by design because of the declarative modeling perspective. On the other hand the process instances can be adaptively changed during runtime to support a flexible change principle. The models are executed within the modeling environment that also includes the runtime environment by the USE tool. The process instance can be changed in the modeling environment during runtime. The adaptive changes do not take place in a distributed Workflow Management System like for example in [15]. The process instance is presented in the same modeling environment known from the design time. The process instances are enriched with execution data like for example timing information.

During the execution of the workflow the time aspect is captured by the workflow plugin and is stored as meta-data of the process instance. After or even during the execution the USE tool provides an interface to state OCL queries for process mining purposes. Thus, USE provides a powerful mechanism to explore properties of the execution data of the workflows.

The rest of this article is structured as follows. In section II the metamodel is introduced. OCL is used to express the semantics of the metamodel elements. In section III the metamodel is applied to model an example workflow with the USE tool and with its abstract syntax. Data and organizational aspects of the workflow are also captured in the model. In section IV the execution of the workflow models is presented. Running processes and activities are shown to the user in a specialized workflow view. Section V discusses related work while section VI concludes the paper.

II. METAMODEL FOR WORKFLOWS

The workflow metamodel is the basis to define the control flow perspective where activities and the causal or temporal interrelations are expressed. In subsection A the metamodel for the control flow perspective is introduced as a UML class diagram enhanced with OCL invariants. Further on, an analysis is given how the workflow patterns [1] are supported. In subsection B the metamodel is enhanced with a data perspective by introducing data flows and queries. Subsection C introduces the metamodel for organizational modeling and connection to the activities.

A. Capturing the Control Flow in the Metamodel

Figure 1 shows the main workflow metamodel. It has been extended in comparison to the version presented in [6] and supports now all 20 workflow patterns [1]. Classes *Activity* and *Iteration* are an integral part of the metamodel. Figure 2 is showing the related object lifecycles as UML state diagrams. OCL invariants are used to define the semantics of the modeling elements. The used states of the state machines of Figure 2 are defined in the enumeration *State* in Figure 1. Calling the operations provided by the classes *Activity*, *Cancel*, *CancelProcess* and *IterationGroup*, object states change according to the state diagrams of Figure 2. The transitions of the state diagrams are implemented by OCL pre- and post conditions. For example the precondition of the *start()* operation requires the object to be in the state *waiting*. Its postcondition consequently assures that the state has changed to *running*. To allow

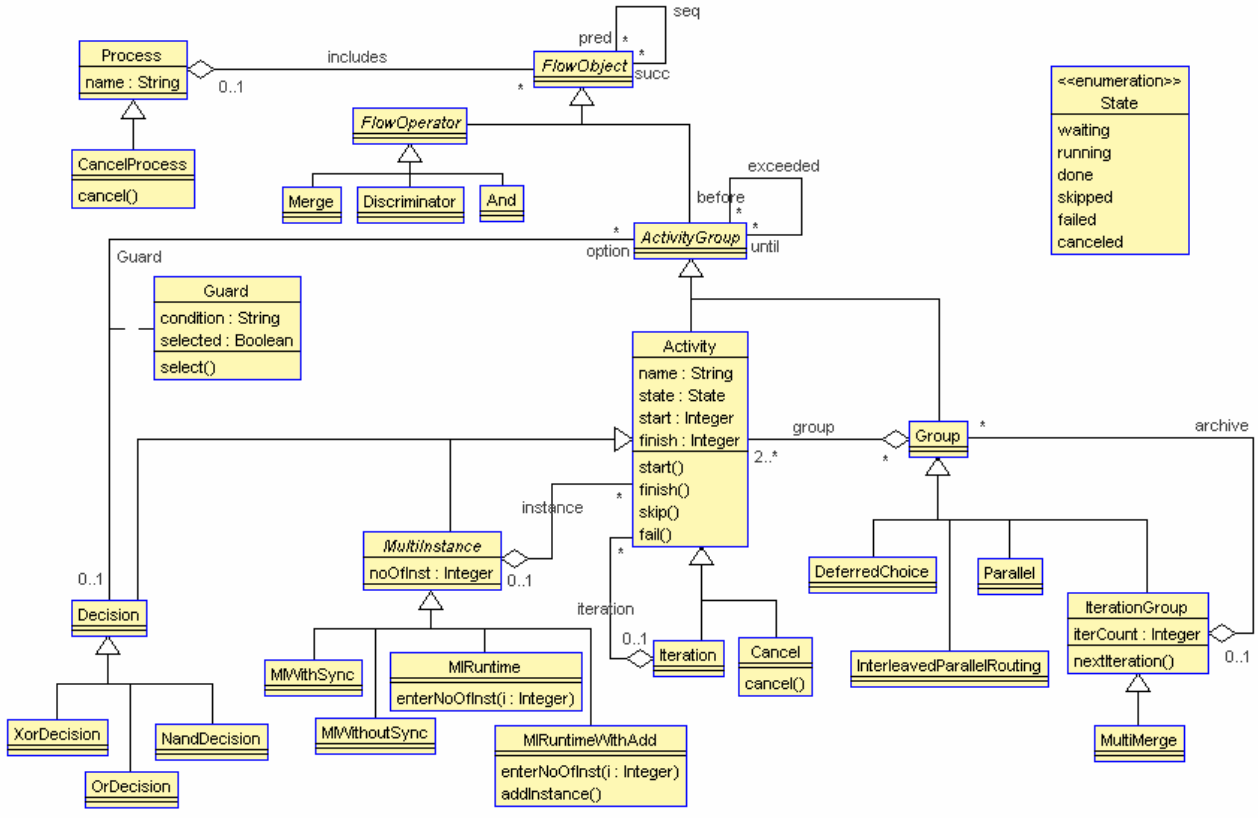

Figure 1: Workflow metamodel

model execution, operations are implemented using the ASSL [9] language. ASSL provides commands for realizing main parts of the UML action semantics. It is a scripting language that is interpreted by USE.

Note that not all operations that change object states have to be assigned to its class. For example the class *IterationGroup* can initiate another iteration with the operation *nextIteration()*. This resets all included activities to the state *waiting* and stores the execution data of the last iteration to a new group linked via the association *archive*. Class *Activity* itself does not directly provide an accessor for resetting its instances' state.

Class *Iteration* state diagram differs from the previous *Activity's* as new iterations can be started after one is finished without resetting the activity. If an *Iteration* object is in the state *running* and the operation *finish* is called new iteration cycles can be started by calling *start* again. The behavior of *Iteration* is described more deeply in [6].

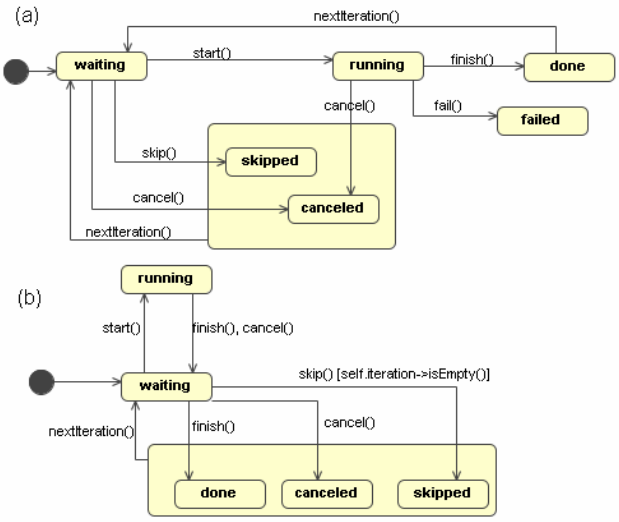

Figure 2: UML state machines to model lifecycles of objects from class (a) *Activity* and (b) *Iteration*

Table 1 lists all workflow control flow patterns (WCP) [1] that are directly or indirectly supported by the metamodel. Patterns that are indirectly supported are explained with example process models. Explanations use the abstract syntax for the workflow models which is provided by the USE tool. OCL invariants that define the semantics of the model elements are omitted in this article. For a deeper introduction of the OCL invariants please refer to [6].

In the metamodel of Figure 1 the class *Activity* is the central part. A process contains several *FlowObjects* that can be *FlowOperators, Groups* or *Activities*. For practical reasons it is sufficient to connect a process object with at least one *FlowObject* that connects all other *FlowObjects* transitively. The operations *getFlowObjects()* and *getActivities()* calculate that transitive closure. They are part of the metamodel but not explicitly shown in Figure 1.

The class *FlowObject* has a reflexive association called *seq* to define the sequence relationship (WCP1). The semantics of the *seq* association is defined in an OCL invariant of the class *Activity*. It says: If an *Activity* is in the state *running*, all its predecessor activities connected via the role *pred* of the association *seq* have to be *done* or *skipped* [6].

TABLE I. WORKFLOW PATTERN ANALYSIS

Workflow Pattern (WCP)	***Expression in the metamodel approach***
1. Sequence	Expressed with the *seq* association and OCL invariant in class *Activity* (see [6])
2. Parallel Split	Expressed with *AndOperator* and *seq* association like pictured in Figure 3
3. Synchronisation	Expressed with *AndOperator* and *seq* association analogue to Figure 3
4. Exclusive Choice	Expressed with *XorDecision* like described in [6]
5. Simple Merge	Expressed with MergeOperator like pictured in Figure 4
6. MultiChoice	Expressed with *OrDecision* like described in [6]
7. Structured Synchronizing Merge	Expressed with *MergeOperator* like pictured in Figure 4
8. MultiMerge	Expressed with *MultiMerge* – all included activities have to have so many iteration cycles as *pred* objects have been executed like pictured in Figure 5
9. Structured Discriminator	Expressed by *Discriminator* – after one *pred* object is executed the *succ* objects can start
10. Arbitrary Cycles	Activities linked arbitrarily with *IterationGroups* like pictured in Figure 6
11. Implicit Termination	Is directly supported – the process is done if all included activities are done, skipped or canceled
12. Multiple Instances (MI) without Synchronization	Expressed with *MIWithoutSync* – the attribute *noOfInst* indicates the number of instances at design time. At runtime the instances are linked by the association *instance*
13. MI with a priori Design-Time Knowledge	Expressed with *MIWithSync* and enriched with an invariant that ensures synchronization compared to WCP12
14. MI with a priori Run-Time Knowledge	Expressed with *MIRuntime* – the attribute noOfInst is determined at runtime by the workflow plugin
15. MI without a priori Run-Time Knowledge	Expressed with *MIRuntimeWithAdd* – has a *addInstance()* operation
16. Deferred Choice	Expressed with *DeferredChoice* – like described in [6]
17. Interleaved Parallel Routing	Expressed with *InterleaveParallelRouting* – like described in [6]
18. Milestone	Expressed by the *exceeded* association like pictured in Figure 7
19. Cancel Task	Expressed with *Cancel* – the cancel interface button is provided by the workflow plugin for these activities
20. Cancel Case	Expressed with *CancelProcess* – all waiting or running activities will be canceled if operation *cancel()* is invoked. The further execution of the process is not possible.

The *seq* association is used in several other contexts, for example with the *FlowOperators* shown as subclasses in Figure 1. The class *And* is used to express WCP2 as demonstrated in Figure 3. An OCL invariant of the class

And defines that all *pred* objects have to be in the state *done* or *skipped* if one of the *succ* activities is allowed to be in the state *running*.

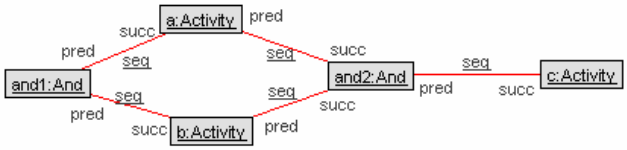

Figure 3: workflow model for WCP2 and WCP3

An OCL invariant for static design time properties is defined which ensures all objects connected with a *FlowOperator* to either be activities or groups. Thus *FlowOperator* cannot be connected to *FlowOperators* by the *seq* association. A similar static property is needed with the WCP8 in which all *pred* objects have to be activities or groups but not flow operators. The expression of that pattern in a workflow model is shown in Figure 5.

Another invariant for static properties of the process models ensures that no *sequence cycles* exist. Those could cause a deadlock during the execution of the workflow model. The invariants are also part of the metamodel. Examples can be found in [6].

In Figure 4 the *MergeOperator* is used in combination with the *XorDecision* activity (WCP4) in a certain structure [12]. It ensures that all *pred* activities are in state *done* or *skipped* before the *succ* activities, or groups of activities, can start. If the *pred* object is a group all included activities have to be *done* or *skipped*. Merges typically appear in such composite structures. This is essential in WCP7 and emphasized by its name: *Structured Synchronizing Merge*. It requires to have a decision directly prior each *Merge*. Decisions are expressed through decision activities that have to be finished before a merge is possible. The non-selected activities and groups are immediately skipped after the decision activity is *done* [6]. With this characteristic the *Merge* can express WCP5 and WCP7.

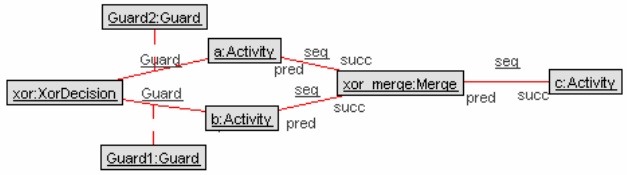

Figure 4: Workflow model for WCP4 and WCP5

The WCP9 is expressed by *Discriminator*. It uses a similar composite structure as the one in Figure 4, but the invariant is somewhat different to the one of *Merge*. It allows *succ* activities connected to the *Discriminator* object to start after one *pred* object is finished.

Figure 5 is showing the WCP8 which is expressed by the *MultiMerge* object. *MultiMerge* is a subclass of *IterationGroup* that allows iterative executions of the included activities one after another by calling the *nextIteration()* operation. To define the semantics of the WCP8, two OCL invariants are used. One says that the number of iterations is not allowed to exceed the number of executed *pred* activities or groups. The other invariant states that all *succ* activities are not allowed to be started before the number of iterations are equal to the number of executed *pred* objects. A group is classified as executed if at least one included activity is *done* and no other activity is *waiting*, *running* or *failed*. As discussed before, an invariant assures the static property that only activities or groups are allowed to be connected as *pred* objects to the *MultiMerge* object.

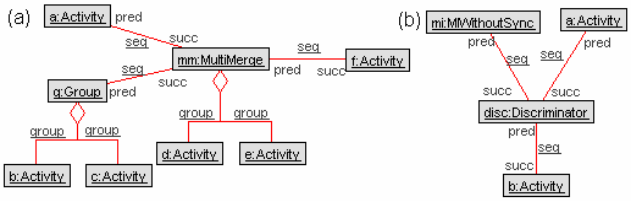

Figure 5: Workflow models for (a) WCP8 and (b) WCP9 and WCP12

In Figure 6(a) the WCP10 is expressed. New iteration cycles can be initiated by calling *nextIteration()* of *IterationGroup* at runtime. An OCL precondition states that all included activities are *skipped*, *done* or *canceled*. The included activities are reset to *waiting* and execution data of the previous iteration is stored through the association *archive*. In this archive every iteration cycle is expressed as a group object that itself is connected to the executed activity instances with the assigned execution data as timestamps in the attributes *start* and *finish*.

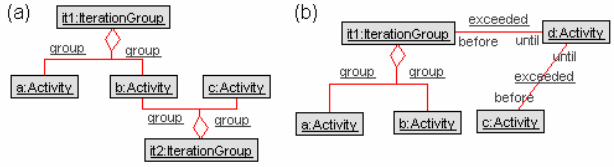

Figure 6: Workflow models for (a) WCP10 and (b) WCP18

In Figure 6(b) WCP18 is expressed by using the *exceeded* association. This association may have side effects on linked activities or groups. If an activity with the role *until* starts, all activities with the role *before* that are not already *started* or *finished* will skip. These side effects are implemented in the ASSL start procedure that is used for the execution of the process models at runtime. If, in the example of Figure 6(b), the activity *d* is started while the activities of group *it1* and the activity *c* are still *waiting* they are skipped. If the current iteration cycle of *it1* is running that iteration can finish but no further iterations can be initiated. This integrity constraint is expressed in the precondition of the *nextIteration()* operation in the class *IterationGroup*.

The *Deferred choice* pattern can also be realized using the *exceeded* association in contrast to the definition that is described in [6]. The activities that are related together with the WCP16 can be mutually linked together with *exceeded* links. If one activity starts all connected ones are skipped

following the semantics of the exceeded association. Thus, the choice is implicitly made.

B. *Integration of Data in the Metamodel*

This subsection discusses the data aspect in the workflow metamodel. USE is a tool that is used for validating UML data models at an early design stage before they are realized in a database system during a system implementation [9]. Data model integration seems to be very promising, because an integrated workflow model can describe data in connection with the control flow properties and vice versa. Further on, these models can be executed in the workflow plugin so that the integrated models can be validated which will be subject of discussion in subsection IV B.

The goal of the data model integration into the workflow metamodel is to specify which data is needed to be captured, edited or read by the user to accomplish its task. Figure 7 demonstrates two options for connecting activities to data objects. The first one is the association *use* that connects activities with objects that represent data queries or creation. Within the attribute *classname* of the class *DataObject* the classname of the queried objects or the object to be created should be specified. The subclasses *DataRead* and *DataEdit* store an OCL selection term, for inquiring data, in the attribute *selection*. These objects are interpreted by the workflow runtime plugin and they are translated in OCL queries. For example a *DataRead* object with the value *'C'* in attribute *classname* and *'z>10'* for attribute *selection* is translated into the OCL query *'C.allInstances()->select(z>10)'*.

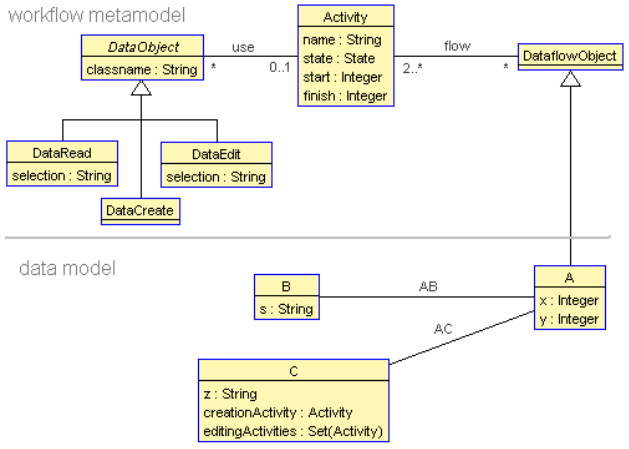

Figure 7: Workflow metamodel with data integration and data model in an UML class diagram

Additionally, there is the possibility to refer to the workflow model from the data model using the attributes *creationActivity* and *editingActivities*. These attributes, see class C in Figure 7, will be recognized by the workflow plugin and if an object is created during the workflow execution the attribute *creationActivity* will be set to the corresponding activity object. If the attribute is not modeled in the class of the created object the backward reference will not be set. In an analogous way the editing of data objects will be recognized by the runtime plugin. If an activity has inquired a data object and attributes have been edited by the user during the activity execution using the workflow plugin the reference to the activity is inserted into the *editingActivities* attribute of the data object.

Having such back references enables the use of data integrity constraints. An example will be given in subsection III C.

Another possibility to integrate data to workflow models is by using the *DataflowObject* and the *flow* association. In the data model an inheritance relationship expresses that objects of a special data class are used to be passed from one activity to another.

Data and workflow modeling uses several different diagram types in this approach. Data modeling employs UML class diagrams while workflow modeling uses UML object diagrams to express the abstract syntax of the workflow language. Table II gives an overview on modeling concepts and their associated diagram types.

TABLE II. USED DIAGRAMS AND PROPERTIES OF THE WORKFLOW- AND DATAMODEL DEVELOPMENT

Development time	***Workflow model with data aspects***	***Data model***
Metamodeling	**Class diagram** pictured in Figure 1 and 7	No metamodel needed (The UML metamodel implemented in USE is used)
Design time	**Object diagram** as a workflow model with faded out workflow-execution data	**Class diagram** – *DataflowObjects* are integrated in the workflow metamodel with inheritance and specially named attributes links to the workflow model like pictured in Figure 7
Runtime	**Object diagram** as a workflow model with enriched workflow-execution data	**Object diagram** – a snapshot of a system state of the data model is generated before or during workflow execution

C. *Integration of Organisational Aspects into the Metamodel*

This subsection is about integration of organizational aspects into the workflow metamodel. In USE an organizational model can be integrated quite easily. Figure 8 shows the organizational metamodel with the integration of the activity class of the workflow metamodel. It expresses the allocation of the *Role* to the *Activity* at design time and the *Person* to *Activity* at runtime.

In the ARIS method and toolset [18] and in several other models like [8] or [7] the organization aspect of an enterprise is modeled hierarchically. Using an organization tree, the root node represents the company as whole. Its subunits are modeled directly beneath it. This dividing into subunits stops at leaves level. Roles are exclusively assigned to organization unit. They can be interpreted as positions in the company [20]. A difference might be that a position is typically assigned to exactly one person. In the organizational metamodel of Figure 8 a person can take over several roles. An example will be given in section III. To achieve the hierarchical structure of the organizational model, the reflexive association *contains* in the metamodel is

used similar to [8] and [7]. Mapping of roles to units are expressed by the association *has*. The correlation of persons to roles is expressed by the *assign* association.

The assignment to activities is achieved by the abstract class *BindingObject* and the association *allocation*. Similar to the ARIS method [18] activities can be assigned to organizational units or rather departments with the association class *alloc_Unit*. The selection of the persons from the department is specified in the attribute *allocType*. Three types are given by the metamodel within the enumeration *AllocationType*. *AnyPerson* selects a idle person of that department to execute the task. *AllRoles* selects for every role assigned to the department a person at runtime to the activity execution. The last value *allPersons* would allocate all persons of the department to the task execution. An example will follow in subsection III C.

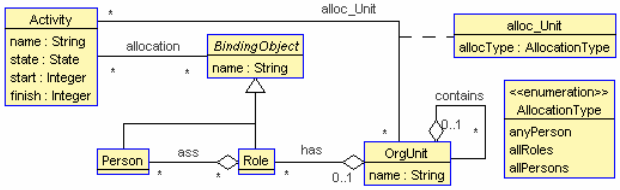

Figure 8: Organizational metamodel with connection to the workflow metamodel

III. Modeling Workflows, Data and Organisational Aspects

In this section we demonstrate how the metamodel is used to model an actual workflow with respect to data and organizational aspects. Firstly the workflow is explained using natural language. Afterwards the metamodels introduced in section II will be applied to create the necessary models. In the last subsection a development process and a tool chain will be presented.

A. The process in natural language

A peri surgical emergency process should be modeled from the arrival in hospital until wake up of the patient.

The process starts with the transportation of the patient. She can either be transported by helicopter or ambulance. For this initial part of the workflow the hospital staff is not responsible to decide what transport type should be taken. Therefore both transportation activities are correlated in a deferred choice relationship [1]. After the patient has arrived at the hospital, she has to be checked whether she has to be operated immediately or if there is time to prepare a normal surgery. This check is done by a doctor at the hospital. Depending on its decision, an immediate or a normal surgery takes place. Afterwards, the patient wakes up which has to be observed by the hospital staff. During this whole process the medication of the patient proceeds and has to be continuously documented. The control flow of the model is extensively pictured in [6].

The model of this paper integrates the data view. Patient data comprises of name (or id), age, disease and urgency of the case. It is determined and entered one after the other in several activities during the process execution. The patient data can be viewed as a dataflow that is handed over from one activity to another one. During transportation name and age of the patient should be determined. After reaching the hospital, an exact diagnosis takes place during the check of the patient. To help a doctor finding the correct diagnosis a list of diseases and respective symptoms could be presented by an information system.

A number of data items is created during the emergency process. For example a nurse has to document if she gives medication to the patient. Kind and amount of medicine are important for the medication protocol. To avoid applying an overdose it can be helpful to present this protocol to her during the execution of the corresponding activity.

B. Modeling workflow and data aspects of the workflow

Initially, needed data has to be modeled in a UML class diagram. As shown in Table II UML class diagrams are used for data modeling at design time. During this step of the design process, a data model may be already validated with USE by creating snapshots as object diagrams and stating OCL queries (see [9]). The data model for the data aspect of the emergency process is shown in Figure 9. Class *PatientData* provides needed data within the attributes. It is a subclass of *DataflowObject* which in turn is a part of the workflow metamodel. Thus, *PatientData* may be used as a dataflow object within the workflow model.

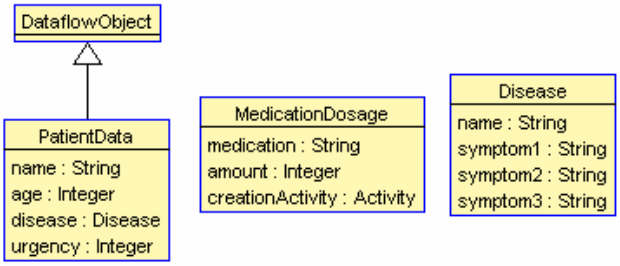

Figure 9: Data model for the emergency workflow model as UML class diagram

Class *Disease* represents possible diseases with their associated symptoms, as stored in the hospital's information system. *PatientData* has an attribute that links to the particular *Disease* which is identified during the initial diagnosis.

Class *MedicationDosage* is used to protocol the medication during an emergency process. For every new medication a new object of the class should be instantiated. Attributes of such newly created object must be filled after the *creationActivity* has been finished. This can be expressed by an OCL invariant similar to the one presented in the following.

Dataflow objects are connected via links of the association *flow* to activities that need the corresponding data object. The integrated model is pictured in Figure 10.
EmergencyProcess is linked with an activity and a group of activities. *AdjustMedication* is independent to any other process fragments. Remaining activities are correlated to the process by calculating the transitive closure as described in subsection II A. After *delivery* is *done*, the patient is

checked. This is expressed with the *seq* link between these objects. Depending on the decision made in *CheckPatientCondition* either the *NormalSurgery* or *EmergencySurgery* take place. *PatientData* is a central data flow object in the model. It is related to the delivery activities and *CheckPatient*. That activity has a *DataRead* object which inquires all diseases from the database of the information system to assist the doctor finding the correct diagnosis. This database is represented by data objects in the UML object diagram. The *AdjustMedication* activity is an *Iteration* that creates a new *MedicationDosage* object with a *DataCreate* object. Additionally, past *MedicationDosage* objects, created by the same activity, are inquired by the *DataRead* object. Past medications are presented to the user so that she can avoid medication overdoses.

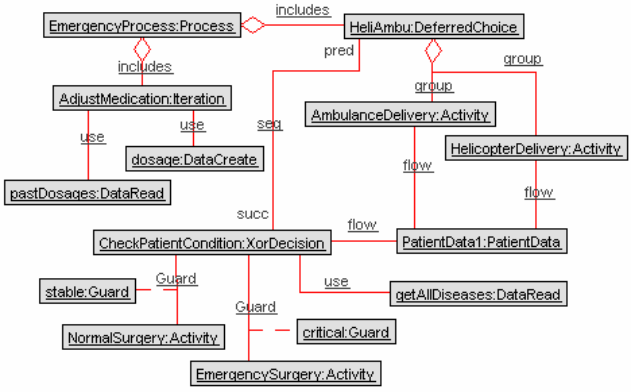

Figure 10: Workflow model with data integration as UML object diagram

The following OCL invariant expresses a data integrity constraint in combination with workflow data. It states: If the transportation of the patient is done the name and age of the patient must be filled.

```
context PatientData inv NameAndAgeFixAfterTransp:
  self.activity->select(state=#done)
  ->includes(name='HelicopterDelivery'
  or name='AmbulanceDelivery') implies
  ( name.isDefined() and age.isDefined() )
```

C. *Modeling the organisational parts*

Modeling the organizational aspect is the core of this subsection. The organizational chart of Figure 11(a) is showing the hierarchical decomposition of the hospital. It is divided into three subunits *FacilityManagement*, *MedicineDepartment* and *Accounting*. Two roles have been introduced and are assigned to the *MedicineDepartment*.

Activities are either assigned to roles or to *OrgUnits* in Figure 11(b). They are assigned to *Persons* at runtime by (ASSL) allocation procedures. The person has to hold the role to accomplish the allocation. Roles and persons are parts of the hierarchical organizational model.

In the model of Figure 11(b) the activities are assigned to corresponding roles. For example *CheckPatientCondition* is executed by a surgeon. *AdjustMedication* is assigned to *MedicineDepartment*. The allocation strategy is indicated by the *anyPerson* object assigned to that link. Thus, any person of that department can execute that task.

Regarding the layout of the models, it is obvious that integrated workflow models are hard to read, because of the number of objects and links that might inevitably tend to overlap. This is not solely a problem of workflow models in USE though its abstract syntax might amplify the effect. But USE offers support by providing a filter mechanism for displaying the desired aspects of a model [9]. Thus, the designer can inquire relevant model elements exclusively for the control flow, organizational or data aspects of the workflow model.

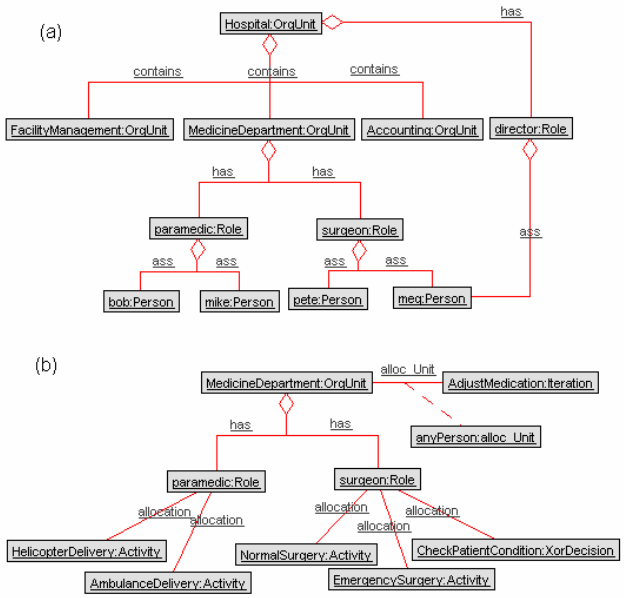

Figure 11: (a) Hierarchical organizational chart (b) Allocation to activities

D. *Development and Validation Process and Tool Chain*

In Figure 12 the development and validation process is pictured.

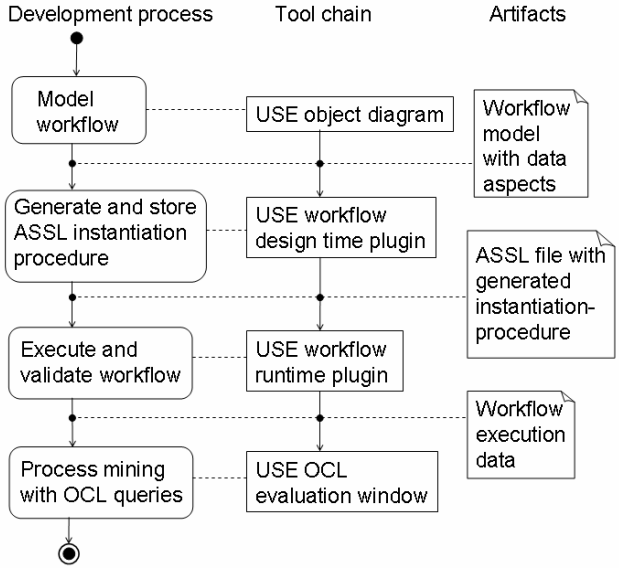

Figure 12: Process and tool chain of model development

The diagrams used in USE and the plugins are listed and related to the activities within the development process. On the left hand side of Figure 12 an activity diagram is pictured. For every activity, a diagram, plugin or tool is linked within the tool chain. The arrows sequenzing the activities and tools represent object flows that are outputs of the previous activity or tool and input for the next activity or tool in the chain. The figure illustrates the order in which the diagrams and plugins are applied in USE.

IV. Executing and Analyzing Workflows

In this section the workflow runtime plugin is presented which provides an environment for workflow instantiation and execution. The plugin presents the workflow instance in an appropriate way to the user so that she can interact with activities work items and related data in an appropriate way. Validation of dynamic control flow properties and related data integration are conducted here. Furthermore, organizational resource aspects can be tested and process mining can be done on the executed processes.

A. *Preparation of the Workflow Execution*

Before the workflow is executed in the workflow plugin, the snapshot of the data model should be prepared in the USE object diagram. In the example workflow of this paper the diseases that should be stored in the hospital information system are created before the process simulation starts.

The process for the workflow instantiation is as follows: The runtime plugin gets the models from the design time plugin as an ASSL file as presented in the tool chain of Figure 12. The user only needs to load the desired ASSL runtime file with the included process instantiation procedures. Afterwards the possible instantiation procedures are listed and the user has to select one, which is consequently instantiated. As a result the process instance appears in the workflow plugin for execution.

B. *Testing dynamical control flow and data aspects with the workflow plugin*

As described in subsection III B the workflow plugin is used to visualize the execution of the workflow models. In the scenario of Figure 13 the process model of Figure 10 is interpreted. The activity *CheckPatientCondition* is started and correlated data is presented to the user within the workflow plugin. The *USE class extent view* is shown on top. It lists all the queried data objects and their attribute values. With the *getAllDiseases:DataRead* object in the workflow model of Figure 10 all the stored diseases are queried. Three were found and are listed in that data window.

The connected *DataFlow* object *PatientData1* was interpreted and is shown within the *USE Object properties* view. The activity *HelicopterDelivery* has already been executed as can be seen by the black colored dot. This represents that activity to be in the state *done*. Activities get particular colors assigned depending on their execution states. Activity states can be changed by the user by clicking on the activity buttons shown on the bottom of the workflow runtime plugin windows.

Values for *name* and *age* have already been entered during the execution of the *AbulanceDelivery* activity, as was assured by the data integrity constraint presented in subsection III B. The attribute *disease* is set to the corresponding *Disease2* object. Moreover, the urgency attribute is entered.

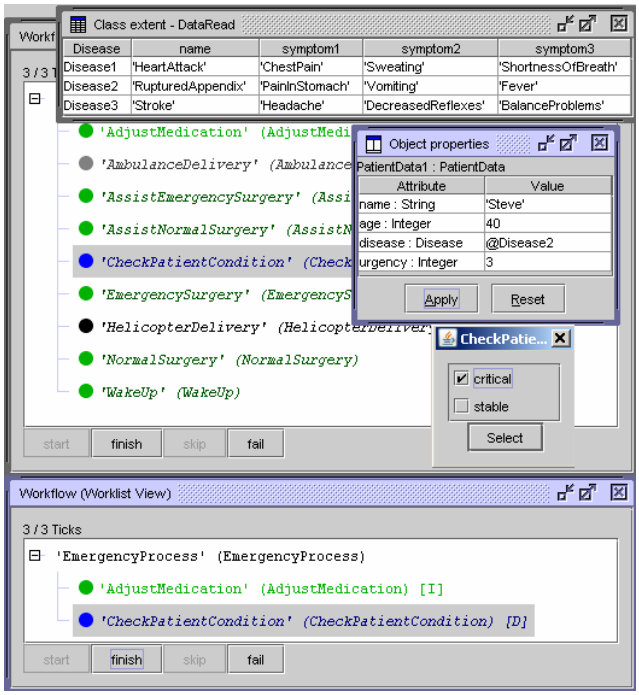

Figure 13: USE workflow runtime plugin with a normal view and the corresponding data presentation and a worklist view

The *CheckPatientCondition* activity is a decision activity. The criterion has to be selected together with the corresponding guard. This is done by the user during the execution of the decision activity within the window shown beneath the *USE Object properties view* in Figure 13. Decision modeling is similar to EPCs, where there is the rule that only activities have the competence to make decisions [11, 5]. UML activity diagrams or YAWL handle decision modeling slightly different. In these languages the choice operators are data driven and automatically executed by the (workflow) system.

The organizational aspects are also tested during the workflow execution. The ASSL allocation procedure which is derived from the design time allocation model of Figure 11(b) tries to find an idle person for executing the activity. If someone is found, then the person is assigned to the activity by the *allocation* link. If none is found the procedure stops with no result. An invariant states that no person can execute two activities in parallel. This should apply for most situations in real life. Nevertheless, the metamodel could be extended to allow activities that can be executed by one person in parallel with certain other activities. This is not considered in the current metamodel.

C. Adaptations during runtime

During the execution of the workflow, model elements can be adaptively changed in the workflow instance. *Adaptation Patterns* [19] are supported on the process instance level. Changes cannot take effect from the instance level back to the process model or "type level" (see [19]).

The user can conduct adaptive changes in the object diagram view of USE. That is already known from the design time shown in Figure 10. Activities, data or organizational aspects can be added, deleted or changed by manipulating the workflow execution data or objects of the data or organizational model.

But the adaptive changes of the workflow instance by deletions and state modifications of activities generate some problems. Certain changes are not reasonable but still allowed by the tool, for example if historical data is deleted in the workflow instance. Furthermore, an activity state change could be conducted although it is not allowed by the life cycle specification of Figure 2. In general *Activity* objects must not be deleted or changed within the object diagram in USE. If an activity should not be executed during the process execution it shall be skipped instead of deleting it in the object diagram. In contrast, adding activities should not cause any problems during the process execution.

Restricting the process execution by adding execution constraints can cause problems, too. But they are recognized by the USE tool as constraint violations. OCL invariants are permanently checked by USE so that constraint violations are instantly displayed by the OCL invariant view [9]. Anyway, the workflow runtime plugin will recognize any changes in the object diagram and it will instantly update its view on the workflow instance model.

To improve the recognition of unreasonable changes that are not already recognized, some pre- and postconditions can be assigned to another operation *adaptiveChange()* added to the class *Process* of the metamodel of Figure 1. Before applying adaptive changes, the operation must be entered in USE [9], then the process instance is modified and afterwards the operation is exited. With OCL the workflow instance state before the adaptive change can be compared with the state after the change so that unreasonable changes can be identified.

Furthermore, an adapted USE object diagram could be implemented as another plugin for USE to provide a user interface that only allows valid adaptive changes.

D. Execution Properties and Process Mining with OCL

During or after the execution of a workflow, its instance data can be evaluated. There are several ways to analyze this data. First of all, the workflow plugin itself is logging the instructions taken from the user. It builds a list in which the call events like *start*, *finish*, *skip*, *cancel* and *fail* are assigned with timestamps and the activity.

Operation calls are logged by an UML sequence diagram. Here side effects of *Activity* operation calls can be easily detected. Changing state of an activity can have side effects on other activities that are related with certain temporal or rather causal relations. A sequence diagram of a process scenario captured in the USE tool can be seen in [6].

The object diagram in USE is showing the workflow instance at runtime in which the execution data is captured. This can also be used for inspection but the data is not presented in an easily readable way. It seems to be more promising to use process mining with OCL queries to get certain properties of a workflow execution. An example OCL term is given in the following. This query lists the activities ordered by elapsed execution time each activity has needed. Any number of further queries can be stated and evaluated in the OCL evaluation window [9].

```
EmergencyProcess.getActivities()->iterate(a:Activity;
acc:Set(Tuple(n:name, t:Integer))=Set{} |
acc->including(Tuple{n:a.name, t:a.finish – a.start}))
->select(t.isDefined())->sortedBy(t)
```

V. Related Work

The ARIS method and toolset is widely used in industry to model business processes with organizational and data aspects [18]. It is a proprietary solution that uses the Event-driven Process Chains (EPCs) and BPMN as workflow languages. The ARIS toolset provides a translation into executable BPEL code which is much more complicated and heavy-weighted compared to the approach presented in this paper.

The Bflow [10] toolbox provides a workflow modeling environment with EPCs. Static aspects of workflow models are instantly checked at design time similar to the static analysis of process models within our approach (see [6]). Bflow also uses a metamodel with EMF and GMF in combination with Eclipse [10]. The models cannot be executed so that dynamic aspects cannot be validated.

Task tree models that represent hierarchy oriented workflow models can be tested and executed within the modeling environment CTTE [13]. This is similar to the method presented here. But CTTE provides less integration of data and organizational aspects and does not allow workflow models to be adaptively changed at runtime.

Declare is a tool for declarative workflow modeling on basis of LTL formulas [2]. The models are executable and the organizational with role-person assignments and data aspects of the workflows are rudimentary captured. The declarative background is similar to the one presented in this paper.

Metamodels are widely used for modeling the interrelations of the workflow models with other diagrams like organizational, data or goal diagrams. Scheer has used metamodels to formally introduce ARIS [18]. Zur Mühlen has evaluated workflow management systems with metamodels [14]. Organizational aspects of workflows are formally modeled in combination with metamodels and OCL in [3] but these models are neither executable nor tested by a UML tool. In the context of organizational modeling workflow resource patterns are identified in [16]. An analysis together with our approach goes beyond the scope of this paper.

Data integration in workflow diagrams is widely done with object flows. Ambiguities of interpreting object flows in UML activity diagrams and BPMN are identified in [4].

These problems are avoided in the approach presented here by having an integrated process and data model within the USE object diagram. The integration of the data views into the workflow models is identified with the workflow data patterns [17].

VI. Summary and Conclusion

In this paper a UML metamodel based approach for workflow modeling is presented. This technique is powerful which is demonstrated by checking it against the workflow patterns. The original 20 workflow patterns can be expressed and some of them can be expressed easier than in popular languages like BPMN or EPCs like for example the *Interleave Parallel Routing Pattern* [6].

Data and organizational aspects can be modeled. An abstract syntax is provided with the UML tool USE for workflow-, data- and organizational modeling. The workflow models can be captured by the developed workflow design time plugin for USE. The model is persistently stored in ASSL-files for later reuse at runtime.

We developed also a workflow runtime plugin which was presented in this paper. It uses the tool realizing parts of the UML action semantics for executing the workflow model based on the ASSL language [9]. The required data objects are presented to the user during the workflow execution. Mutual dependencies can be specified between data and the control flow specification. Such constraints are observed by USE and missing data entries are identified during workflow execution. The user is directed to the corresponding objects that violate the OCL invariants.

The workflow instances can be analyzed and adaptively changed at runtime. The same applies to the snapshots of the organizational and data model. Time aspects are captured by the workflow plugin and are stored during workflow execution. Thereafter, this data can be analyzed in a log window of the workflow plugin, by a UML sequence diagram or by OCL mining queries. So, important properties of the executed process instances can be discovered.

References

[1] W.M.P. van der Aalst, A.H.M. ter Hofstede, and Kiepuszewski, "Workflow Patterns," *Distributed and Parallel Databases*, 14(3):5-51, 2003.

[2] W.M.P. van der Aalst, M. Pesic, and H. Schonenberg, Declarative workflows: Balancing between flexibility and support. Computer Science - Research and Development, 23(2):99–113, 2009.

[3] W.M.P. van der Aalst, and A. Kumar, "Team-Enabled Workflow Management Systems," Data and Knowledge Engineering, 38(3):335-363, 2001.

[4] J. Brüning, and P. Forbrig "Behaviour of flow operators connected with object flows in workflow specifications," 7th International Conference on Perspectives in Business Informatics Research (BIR2008), University of Gdansk, 2008.

[5] J. Brüning, and P. Forbrig, "Modellierung von Entscheidungen und Interpretation von Entscheidungsoperatoren in einem WfMS", EPK 2009 Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten, Berlin, CEUR-WS 554, 2009.

[6] J. Brüning, M. Gogolla, and P. Forbrig, "Modeling and formally checking workflow properties using UML and OCL," 9th International Conference on Perspectives in Business Informatics Research (BIR2010), LNBIP vol. 64, Springer, 2010.

[7] H.-E. Eriksson, and M. Penker, "Business Modeling with UML: Business Patterns at Work," Wiley, 2000.

[8] M. Fowler, "Analysis Patterns: Organization Structures (Accountability)," http://martinfowler.com/apsupp/accountability.pdf (visited: 02/28/2011)

[9] M. Gogolla, F. Büttner, and M. Richters, "USE: A UML-Based Specification Environment for Validating UML and OCL," *Science of Computer Programming*, 69:27-34, 2007.

[10] S. Kühne, H. Kern, V. Gruhn, and R. Laue, "Business process modeling with continuous validation," Journal of Software Maintenance and Evolution: Research and Practice, Volume 22, Issue 6-7, pages 547–566, 2010. DOI: 10.1002/smr.517

[11] G. Keller, M. Nüttgens, and A.-W. Scheer, "Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK)". *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, Heft 89, Saarbrücken, 1992.

[12] B. Kiepuszewski, A.H.M. ter Hofstede, and C.J. Bussler, "On Structured Workflow Modelling," 12th International Conference on Advanced Information Systems Engineering (CAiSE2000), Stockholm, LNCS vol. 1789:431-445, 2000.

[13] G. Mori, F. Paterno, and C. Santoro, "CTTE: Support for Developing and Analyzing Task Models for Interactive System Design," IEEE Transactions on Software Engineering, 2002, pp.797-813.

[14] M. zur Mühlen, "Evaluation of Workflow Management Systems Using Meta Models", In: R. Sprague Jr, editor, 32nd Annual Hawaii International Conference on Systems Sciences, Wailea, Hawaii, USA, 1999.

[15] P. Dadam, and M. Reichert, "The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support - Challenges and Achievements," Computer Science - Research and Development, Springer. Vol. 23, No. 2, pp. 81-97, 2009.

[16] N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst, "Workflow Resource Patterns," BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004.

[17] N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst, "Workflow Data Patterns," QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.

[18] A.-W. Scheer, " ARIS: Business Process Modeling," Springer, 2000.

[19] B. Weber, Rinderle, S., and Reichert M., "Process Change Patterns (Aktuelles Schlagwort)," EMISA Forum, 27(2):45-51, 2007.

[20] M. Weske, "Business Process Management," Springer, 2007.

[21] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russel, "Pattern-based Analysis of the Control-Flow Perspective of UML Activity Diagrams," 24th International Conference on Conceptual Modeling (ER2005), LNCS vol. 3716, Springer, 2006.

[22] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede, "Pattern-Based Analysis of BPEL4WS, " QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.