Modeling Concepts for Workflow Specification

Markus Kradolfer, Andreas Geppert

This paper describes modeling concepts for the specification of processing entities and workflows and a workflow execution model for the TRAMs[1] project.

1 Introduction

A *workflow* is either a basic work step, called *activity*, or a *complex workflow* that consists of one or more workflows. Basically, the complexity of activities is arbitrary. This means, an activity can be a trivial task such as a simple read access to a database or it can be a complex task, such as the determination of the credit-worthiness of a company. A workflow x that is contained in another workflow y is called *subworkflow* of y and workflow y is called *superworkflow* of x. Workflows that have the same superworkflow are called *sibling workflows*. A workflow that is not part of another workflow is called *top-level workflow*.

Activities are carried out by *processing entities* which can range from humans to arbitrary software systems. Each activity and complex workflow is an instance of a certain *workflow type*. More precisely, an activity is an instance of an *activity type* and a complex workflow is an instance of a *complex workflow type*. From a workflow type any number of instances (workflows) can be created. A workflow type definition (synonymous to workflow specification) is a formal

1. TRAMs (**TR**ansactions and **A**ctive Mechanism**s** for Workflow Management) is a joint project of the University of Zurich and the ETH Zurich

description of the common aspects for a set of workflows which are relevant for their execution.

As shown in Figure 1, a *process-centered environment* consists of a set of processing entities and a *workflow management system (WFMS)*. A WFMS is a software system that provides for the specification, enactment and monitoring of workflows. Workflow enactment does not mean that the WFMS itself carries out the activities workflows consist of. Rather the WFMS ensures that the processing entities execute the activities in a coordinated way according to the corresponding workflow specification. A WFMS consists of two major components:

- *specification component*: provides for the specification, persistent storage and management of information related to workflow types and processing entities.
- *enactment component*: provides for the enactment of workflows according to the workflow specifications created with the specification component.

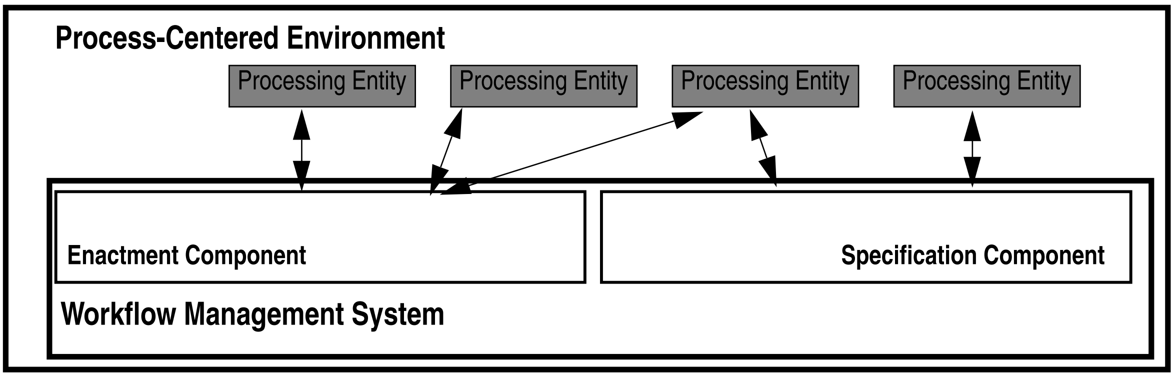

Figure 1: Logical Architecture of a Process-Centered Environment

The specification component offers one or more *specification languages*, which are formal languages that offer constructs to capture the information necessary to enact workflows. A specification language has an underlying *model*[2], which defines the modeling concepts the language offers through its constructs in a high level way.

In this paper we define, independent of concrete specification languages, a *workflow model*, an *organizational model*, and a *workflow execution model*:

- the workflow model defines a set of modeling concepts that can be used to specify workflows,
- the organizational model defines a set of modeling concepts for the specification of processing entities, their properties, and the relationships that exist between them,
- the workflow execution model defines how workflows are executed (execution semantics).

The remainder of the paper is organized as follows. First, an example workflow is presented in Chapter 2. Thereafter, basic assumptions and requirements are formulated in Chapter 3. Chapter 4 defines the workflow, organizational and workflow execution model.

2. The different meanings of the term model, especially in the software engineering field and the database technology field, can cause confusion. In this paper we use the term in the sense it is used in the database technology field. The term equivalent to workflow model would therefore be workflow meta-model in the software engineering field, and the workflow specification would be called workflow model.

2 Example Workflow Type: Health Insurance Claim

In this chapter, we present an example workflow type which is a modified version of the example in [GKT95]. The application scenario is the processing of a health insurance claim (HIC) in an insurance company. In this chapter we only give an informal overview of the workflow type. In Chapter 4, various aspects of the workflow type are defined in greater detail and in a more formal way.

In Figure 2 the workflow type is represented by a dependency graph which defines some aspects of the control flow. The graph consists of oval nodes representing activities and black arrows connecting the nodes. An arrow pointing from an activity x to an activity y means that the start of y depends somehow on one or more properties (e.g. execution state, results produced) of activity x. To each activity, a processing entity is attached.

Once a HIC is received, a clerk creates an electronic dossier containing the diagnosis, the treatments, and the costs, and if an insurance policy exists, a reference to the entry in the insurance company database (enter case). If no insurance policy exists for the applicant or if the cost is less then 150 Swiss Francs, the HIC is rejected (reject HIC), i.e. a notification of the rejection is printed. Otherwise, if an insurance policy exists and the cost is below 500 Swiss Francs, the HIC is accepted (accept HIC), i.e. an entry in the payment history of the applicant is made and a payment check is printed. If an insurance policy exists and the cost is above 500 Swiss Francs, several checks are done. First, an expert system checks whether some of the treatments are contained in a blacklist (check blacklist) in which case the HIC will be rejected (reject HIC). If the expert system has no complaints about the treatment, two further checks are done in parallel. An expert system controls weather the treatment suits the diagnosis (check compatibility) and a medical specialist checks if the treatment makes sense with respect to the patient's history (check history). If one or both of these two checks fail, the HIC is rejected (reject HIC). Otherwise, the HIC is accepted (accept HIC). Finally, a law specifies that the insurance company has to react (either positively or negatively) at the latest after six weeks. Therefore, we assume that if the HIC has not been rejected or accepted after six weeks, the claim is automatically accepted.

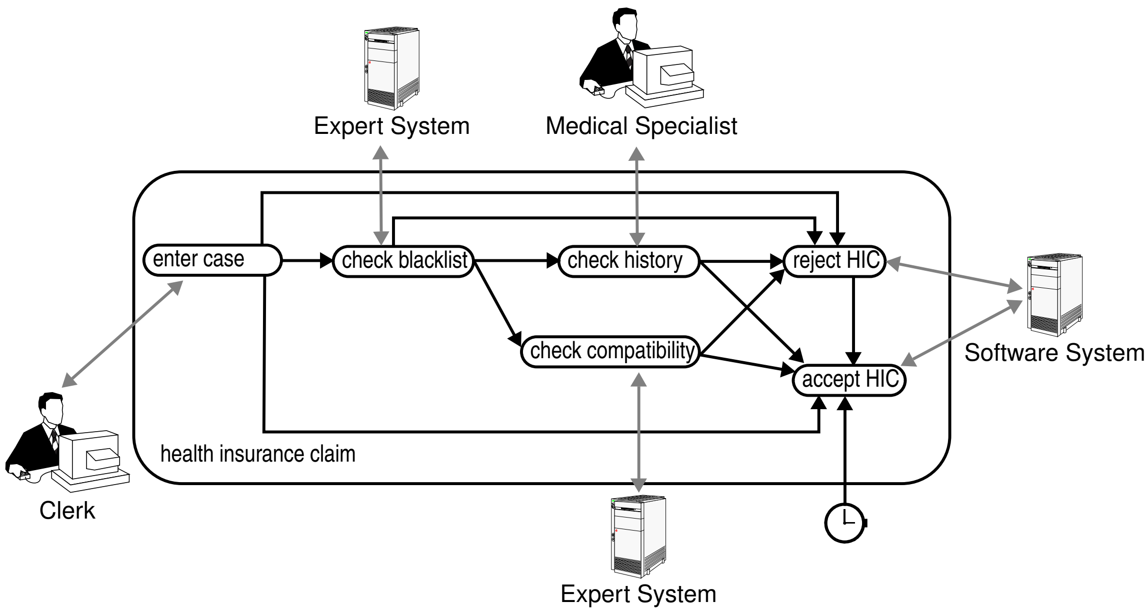

Figure 2: Sample Workflow: Health Insurance Claim Processing

3 Assumptions and Requirements

In this chapter we first describe the type of process-centered environments for which the modeling concepts defined in this paper are intended for. Thereafter, requirements on the workflow model, on the organizational model, and on the workflow execution model are identified.

3.1 Process-Centered Environment

The process-centered environment (PCE) consists of processing entities and a workflow management system. The processing entities provide activity types. If a processing entity provides an activity type AT1 this means it has the ability to carry out activities of type AT1. An activity type can potentially be offered by more than one processing entity.

The PCE is dynamic in that

- processing entities may leave and enter the environment
- the set of activity types a processing entity provides may change over time

It is assumed that processing entities maintain their own data, called *production data,* locally. Thus, the WFMS is not used as a data management system. Still, the workflow management system has to store and maintain data relating to workflows in order to be able to enact them. This data is called *control data.* Control data can include references to production data maintained by processing entities.

Furthermore, the processing entities are heterogeneous, distributed and autonomous:

- *heterogeneous*: the processing entities have heterogeneous programming interfaces and do not use a uniform data model for production data modeling. Additionally, processing entities differ in the support of transactions.
- *distributed*: the processing entities may be distributed over wide area networks.
- *autonomous*: the processing entities are autonomous in various respects:
  - *autonomous regarding the execution of activities,* i.e. processing entities retain complete control over the execution of activities. In particular, they are autonomous in the way activities are carried out. Processing entities are regarded as blackbox components whose internal mode of operation and degree of automation is not known to the outside.
  - *autonomous regarding communication with the WFMS*, in that processing entities do not share information about activities with the WFMS until the activities are finished. It may even be the case that processing entities are not connected to the WFMS all the time. It could well be the case that a processing entity receives requests to carry out a set of activities, then disconnects from the WFMS, carries out the activities in full isolation and reconnects to the WFMS after having completed the activities.

In order to address distribution and heterogeneity of processing entities, we assume that the processing entities are integrated into an environment similar to CORBA. Thus, the programming interfaces of the processing entities are described in a uniform language (e.g. Interface Definition Language in the case of CORBA).

3.2 Requirements on the Workflow Model

Basically, which information has to be definable using a workflow model depends on the requirements stemming from the application domain. Stated otherwise, the completeness of a workflow model is relative to the application. For example, in one application domain data flows do not have to be modeled, or in another application domain, deadlines play an important role, such that for each workflow a deadline has to be defined and an appropriate reaction in case the deadline is reached. In a third application domain, costs are a central issue, such that the costs of activities and processing entities per time unit have to be modeled in order to enable the calculation of the costs of workflows. Because constructing a workflow model that satisfies all requirements from all possible application domains is hardly feasible, there is no such thing as a "complete" workflow model. Since the focus in TRAMs, as stated in the previous section, is not on a specific application domain, a set of minimal requirements common to many application domains are identified. We structure the requirements according to the four major perspectives workflows can be looked at, as described in [CKO92]:

- functional/structural perspective: **Which** workflows have to be performed and how are they structured?
- informational perspective: Which **data** is produced and consumed by workflows and how data flows among workflows (**data flow**)?
- behavioral perspective: **When** have the workflows have to be performed (control flow)?
- organizational perspective: **Who** has to carry out the activities?

Functional/ Structural Perspective

In the functional/structural perspective, the focus is on which workflows are performed and how they are structured. A workflow model should support:

- the specification of basic work steps: activity types
- the specification of complex workflows: complex workflow types
- a concept to locally group a subset of the subworkflows of a complex workflow.

Informational Perspective

In the informational perspective, the focus is on the data consumed and produced by workflows and on the flow of data between workflows.

A workflow can consume and produce data during execution. In order to enable the specification of data flows between workflows and the specification of control flow (e.g. conditional executions of workflows depending on output data produced by other workflows), the data consumed and produced by a workflow has to be declared explicitly by defining input and output parameters for them. For obvious reasons, the parameters should be typed. Therefore a type definition facility has to be provided. The kind of data types that should be definable depends on the application domain.

Data flows among workflows should be definable. This means that the workflow model should support the specification of associations between output parameters and input parameters of workflows, such that it is possible to define that one workflow consumes data that was produced previously by another workflow.

Behavioral Perspective

In the behavioral perspective, the focus is on *when* workflows are executed. The specification of the following facts should be supported:

- conditional execution of workflows, depending on properties of other workflows (e.g. result produced or execution state) or external variables that are not modified by any workflow (e.g. time)
- temporal constraints such as deadlines workflows have to meet

- a variable number (whose actual value will not be determined before execution time) of workflows of a certain type that have to be executed sequentially (iteration) or that may be executed in parallel

Organizational Perspective

In the organizational perspective, the focus is on *who* (which processing entities) carries out the activities workflows consists of. Starting from the fact that one activity type may be provided by more than one processing entity, one of the major requirements on a workflow model is to provide powerful concepts to define to which processing entities activities will be assigned for execution. The assignment of an activity can be done on the basis of e.g.

- individual properties of processing entities (e.g. capabilities)
- relationships between processing entities and organizational entities (such as other processing entities, groups or roles) or other modeling entities such as activity types (e.g. which activity types are provided by which processing entities).
- properties of other workflows including previous assignments of processing entities to activities. For example, it might be required that the processing entity that will carry out an activity x is different from the processing entity that carried out the activity preceeding activity x.
- variables that are not modified by any workflows (e.g. time)

3.3 Requirements on the Organizational Model

Proceeding from the requirements listed in the above paragraph, an organizational model has to support the definition of:

- processing entities (including their properties)
- additional organizational entities such as groups, departments, roles, etc.
- relationships between the above entities, e.g. supervisor_of between processing entities, manager_of between processing entities and organizational units such as groups or departments

The kinds of organizational entities and relationships among them differ widely from organization to organization. Therefore, the organizational model should allow the definition of new user-defined entity types and relationship types.

4 Workflow, Organizational, and Workflow Execution Model

In this section, the workflow model, the organizational model, and the workflow execution model are described. The workflow model defines the concepts for workflow specification, the organizational model the ones for the modeling of processing entities. The workflow execution model defines how workflows are executed.

Subsequently, the workflow model as well as the organizational model is defined using the object model, terminology, and the class diagrams of the Unified Modeling Language (UML) [RSC97]. Figure 3 shows the main elements of the class diagrams. Each of the two models defines a set of classes, the attributes of the objects, structural relationships called associations that exist between the objects of different classes, and inheritance relationships that exist between the classes. The classes are numbered ascendingly in the order they are introduced.

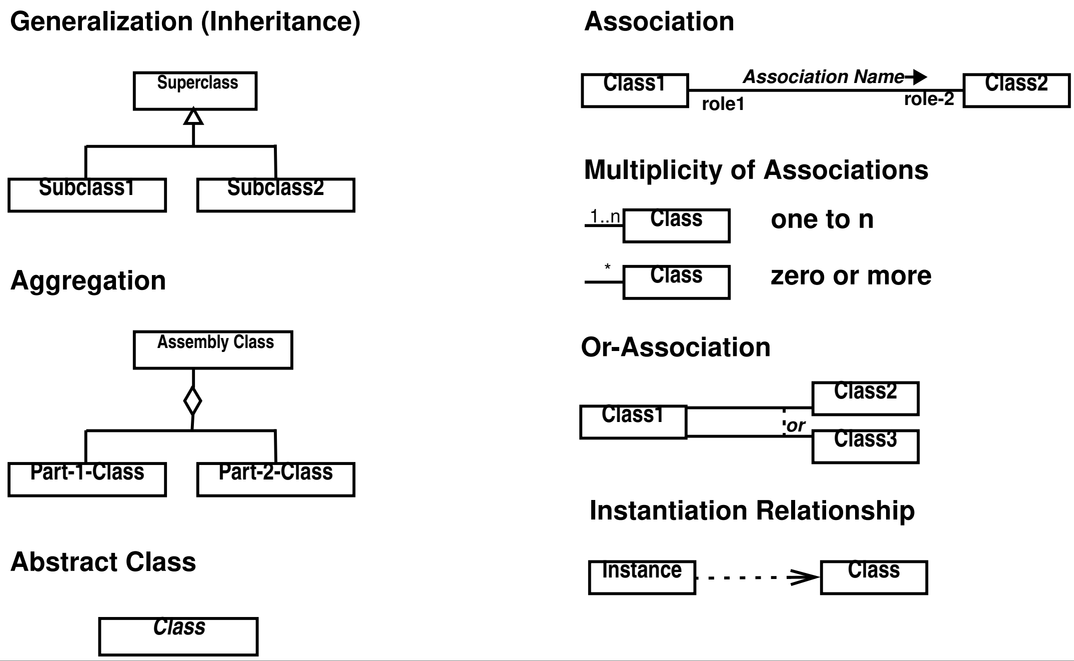

Figure 3: Class Diagrams of the Unified Notation [RSC97]

Section 4.1 describes the organizational model. In Section 4.2 the core workflow model is defined, followed by the description of the execution model in Section 4.3. Section 4.4 extends the core workflow model by two constructs which allow the specification of sequential and parallel execution of a variable number of workflows. In Section 4.5, the workflow model is extended to support the specification of deadlines for workflows. Finally, the definition of inheritance relationships between complex workflow types is discussed in Section 4.6.

4.1 The Organizational Model

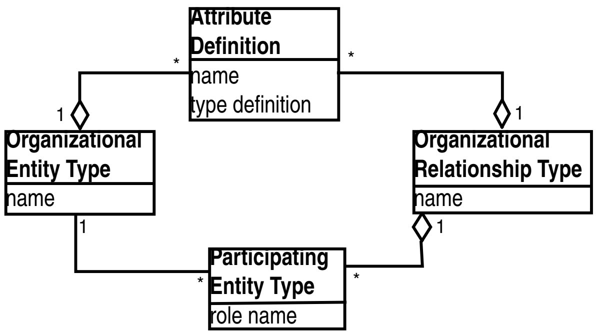

Figure 4: Main Concepts of the Organizational Model

As illustrated in Figure 4, the organizational model allows to define *organizational entity types* and *organizational relationship types*. An organizational entity type defines a name and a set of attributes. Similarly, an organizational relationship type also defines a name and a set of attributes. In addition, it defines a set of organizational entity types that take part in the relationship.

The organizational model includes a set of predefined organizational entity types and organizational relationship types. In Figure 5 the organizational entity types are shown as classes, the organizational relationship types as associations.

A *processing entity* represents an individual processing entity and has a set of *capabilities*. Furthermore, it can be a member of one or more *organizational units*. An organizational unit can be part of one or more other organizational units. Processing entities can play *roles*, which represent scopes of functions. It is important to note that all the associations of Figure 5 can change during runtime. This means that e.g. a processing entity can loose or gain capabilities or change its membership to organizational units in the course of time.

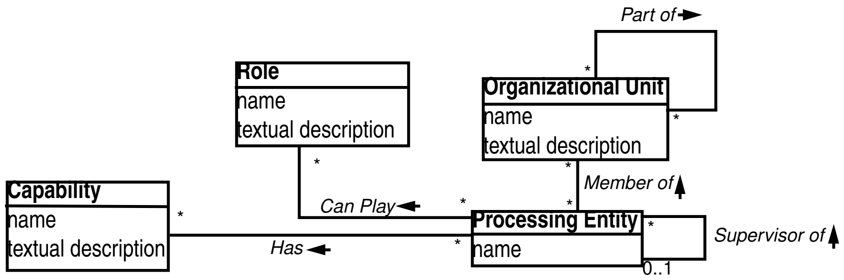

Figure 5: The TRAMs Core Workflow Model Classes of the Organizational Perspective

HIC Workflow Type

Figure 6 presents an organizational description for the HIC example of Chapter 2 using a subset of the predefined organizational entity and relationship types.

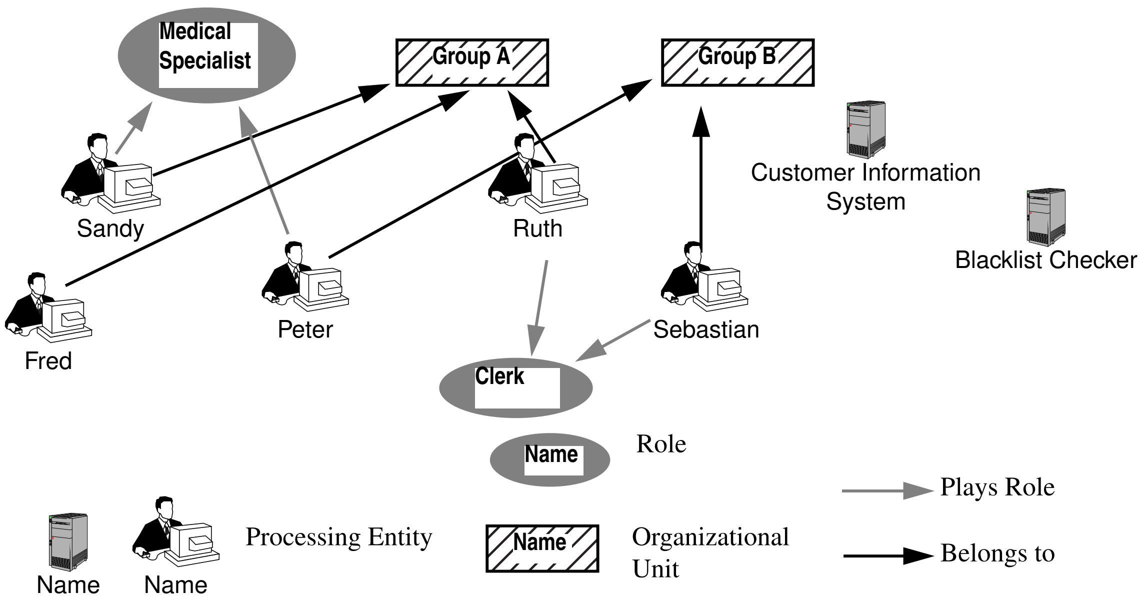

Figure 6: Organizational Modeling for the HIC Example

The classes that define the organizational model are defined as follows:

Definition 1 Organizational Entity Type

Attributes:

- name of the entity type

Aggregations:

- attribute definitions: a set of objects of class Attribute Definition

Definition 2 Organizational Relationship Type

Attributes:

- name of the relationship type

Aggregations:

- attribute definitions: a set of objects of class Attribute Definition
- the organizational entity types that participate in the relationship type: a set of object of class Participating Relationship Type

Definition 3 Attribute Definition

Attributes:

- name of the attribute
- type of the attribute: definition in the Object Database Management Group Object Definition Language (ODMG ODL) [Cat94]

Definition 4 Participating Relationship Type

Attributes:

- role name

Association:

- organizational entity type that takes part in the relationship type: a reference to an object of class Organizational Entity Type

Definition 5 Processing Entity (Instance of class Organizational Entity Type)

Attributes:

- name

Associations:

- supervisor_of associations:
  - supervisor of the processing entity: reference to an object of class Processing Entity
  - subordinates of the processing entity: references to a set of objects of class Processing Entity
- the organizational units the processing entity is part of: references to a set of objects of class Organizational Unit
- the roles the processing entity can play: references to a set of objects of the class Role

Definition 6 Capability (Instance of class Organizational Entity Type)

Attribute:

- name
- textual description

Associations:

- processing entities that provide this capability: references to a set of objects of class Processing Entity

Definition 7 Role (Instance of class Organizational Entity Type)

Attribute:

- name
- textual description

Associations:

- processing entities who can play this role: references to a set of objects of class Processing Entity

4.2 The Core Workflow Model

In this section, the core workflow model is defined. The classes that define the workflow model are categorized into *specification level classes* and *execution level classes*. Figure 7 shows the main classes of the workflow model and their categorization. The classes of the specification level are Activity Type and Complex Workflow Type. Objects of these classes represent the entities that can be defined using the specification component. Thus, in TRAMs the specification repository contains specifications of activity types, complex workflow types and processing entities. Objects of the execution level classes (Activity, and Complex Workflow) represent entities that exist in the workflow execution environment: activities and complex workflows.

There exist instantiation relationships between objects of the specification level classes and objects of the execution level classes. For example an object of the class Activity is an instance of an object of the class Activity Type.

The main classes the core workflow model consists of are shown in Figure 7. In Figure 8 and Figure 9 additional classes are introduced which together form the core workflow model. The definition of the workflow model is divided into the four sections 4.2.1 through 4.2.4, according to the four workflow perspectives already introduced in Section 3.2. Each section includes

- a class diagram of the classes relevant for the perspective the section deals with
- a description of the relevant attributes of objects of these classes and of the associations that exist between the objects of these classes
- graphical constructs for the visualization of some aspects of objects of the classes introduced in the section
- a graphical and/or textual description of the HIC example workflow type

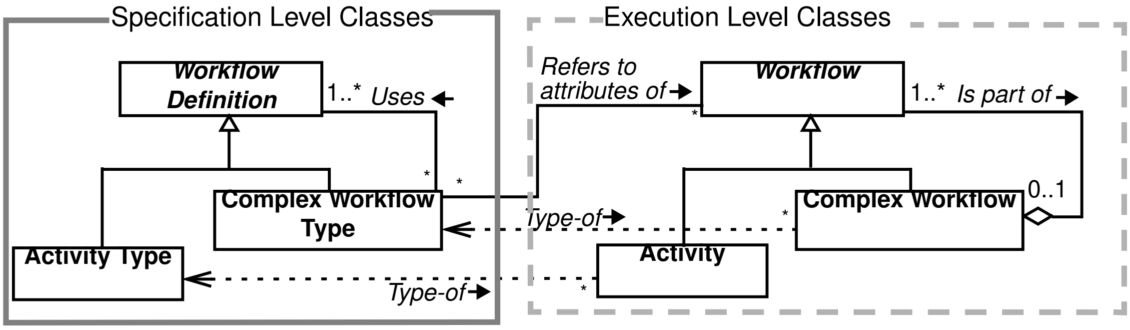

Figure 7: The Main Classes of the Workflow Model

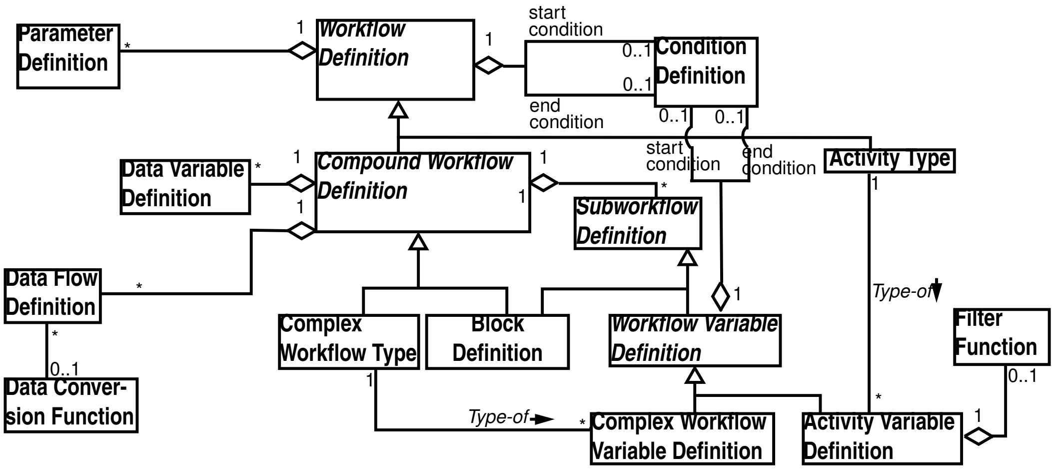

Figure 8: The TRAMs Core Workflow Model: Specification Level Classes

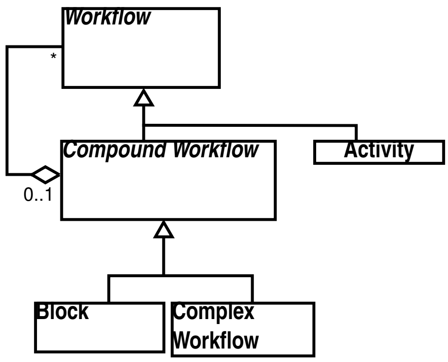

Figure 9: The TRAMs Core Workflow Model: Execution Level Classes

4.2.1 Functional/ Structural Perspective

Figure 10 illustrates the workflow constructs relevant in the functional/structural perspective. A workflow definition is either an *activity type*, representing a basic workstep, or a *compound workflow definition*. A compound workflow definition is either a *complex workflow type* or a *block definition* and consists of a set of subworkflow definitions. A *subworkflow definition* is either a block definition or a *workflow variable definition.*

.

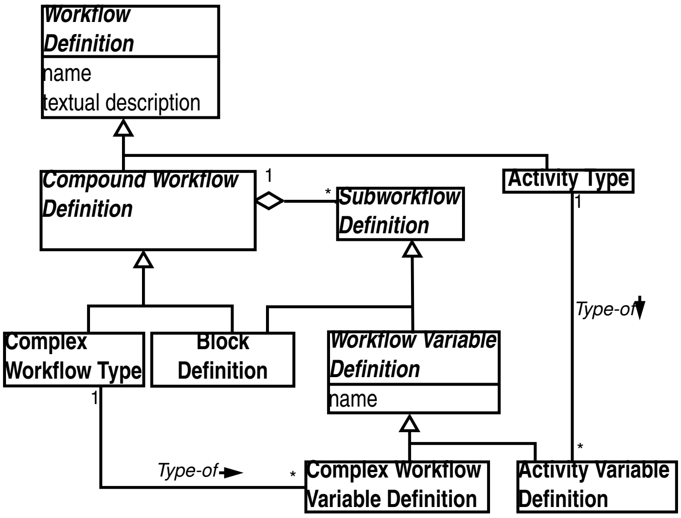

Figure 10: The TRAMs Core Workflow Model Classes of the Functional/Structural Perspective

The block definition construct is supported in order to provide the possibility to locally structure workflow definitions and to specify the sequential or parallel execution of a variable number of workflows (see Section 4.4). A similar construct is provided in the modeling language of FlowMark [LA94].

Workflow definitions, as well as workflow variable definitions include a textual name (identifier). The names of workflow types (activity types or complex workflow types) have to be unique. A subworkflow definition name that is part of a compound workflow definition cwd1 has to be different from the name of cwd1 and from the names of the other subworkflow definitions included in cwd1.

The classes relevant in the functional/structural perspective (shown in Figure 10) are defined as follows:

Definition 8 Workflow Definition (abstract class)

Attributes:

- name
- textual description of what the workflow does

Definition 9 Activity Type (subclass of Workflow Definition)

-

Definition 10 Compound Workflow Definition (abstract class, subclass of Workflow Definition)

Aggregations:

- definition of subworkflows: a set of objects of class Subworkflow Definition

Definition 11 Complex Workflow Type (subclass of Compound Workflow Definition)

-

Definition 12 Block Definition (subclass of Compound Workflow Definition and Subworkflow Definition)

-

Definition 13 Subworkflow Definition (abstract class)

Associations:

- compound workflow definition the subworkflow definition is part of: reference to an object of class Compound Workflow Definition

Definition 14 Workflow Variable Definition

Attributes:

- variable name

Definition 15 Activity Variable Definition (subclass of Workflow Variable Definition)

Associations:

- variable type: reference to one object of class Activity Type

Definition 16 Complex Workflow Variable Definition (subclass of Workflow Variable Definition)

Associations:

- variable type: reference to one object of class Complex Workflow Type

Graphical Representation

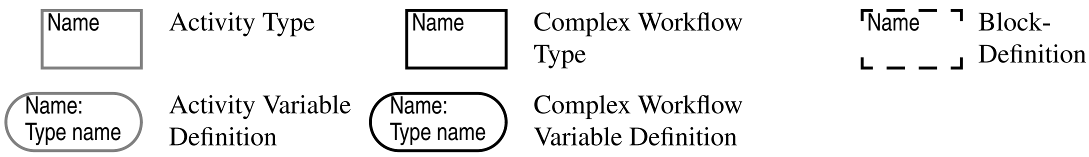

Figure 11: Graphical Representation of Workflow Model Constructs

Figure 11 shows the graphical representation of the workflow model constructs introduced in this section. Since certain workflow constructs have to or can be defined inside others, containment relationships exist between them. Containment relationships can be visualized by trees, as shown in Figure 12 b) such that workflow constructs are represented by nodes. The workflow construct which is represented by a node n contains the workflow constructs represented by the children of n. Figure 12 a) illustrates another way to visualize containment relationships. A workflow construct x that is contained in another workflow construct y is represented by a graphical symbol that is drawn inside the graphical symbol representing y.

The Functional/Structural Perspective of the HIC Workflow Type

Figure 12 shows the structural perspective of the HIC Workflow Type using the graphical elements defined above.

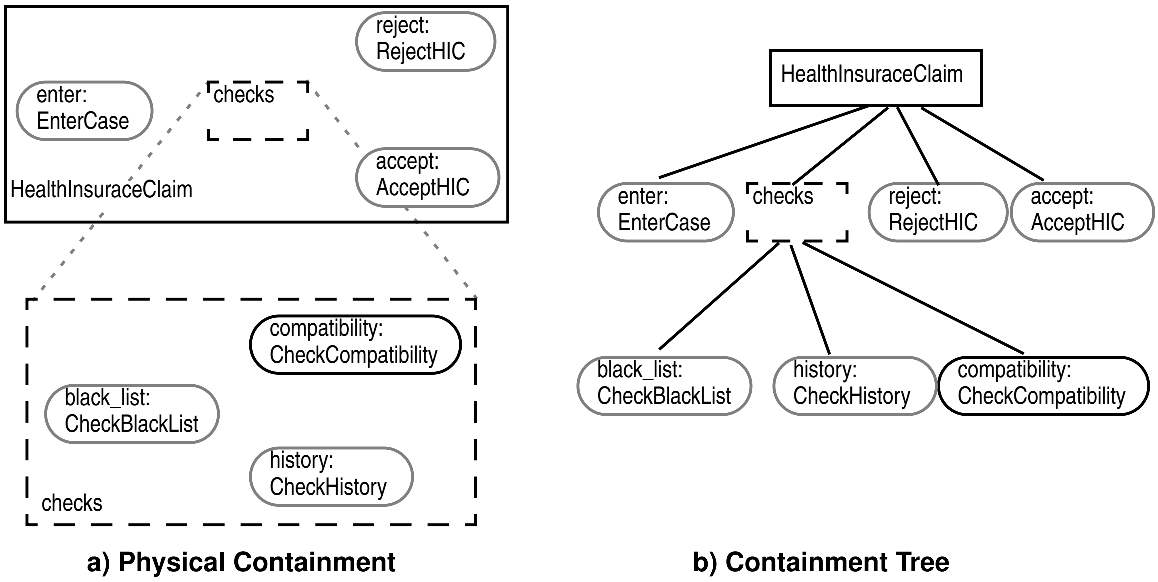

Figure 12: Structural/Functional Perspective: Health Insurance Claim Processing

4.2.2 Informational Perspective

In the informational perspective, the focus is on the data consumed and produced by workflows and on the data flow between workflows.

Parameters

For each complex workflow type, activity type and block definition, in, out, and inout parameters may be defined. The parameters are defined using ODL.

It is important to note that a workflow may consume and produce data in addition to its defined parameters. Thus, data flows can occur *implicitly* in that a workflow reads a certain data item which has been written by another workflow. But since one of the main goals of workflow management is to separate the basic work steps from the dependencies that exist between them (including data flows), implicit data flows should be avoided.

Data Variables

Data variables can be defined by complex workflow types and block definitions. Data variable definitions are graphically represented as grey circles (illustrated in Figure 13).

Data Flows

A data flow definition relates a *data source* to a *data sink*. A data source or sink is either a parameter or a data variable. Data flows can be defined by compound workflow definitions (complex workflow types and block definitions). As shown in Figure 13, data flow definitions are graphically represented by gray *data flow arrows*, connecting graphical symbols which represent data variable definitions, complex workflow types, and subworkflow definitions. A data flow arrow pointing from a graphical symbol x to a graphical symbol y means that there exist one or more data flow definitions with the following properties. The source of the data flow definitions is either the workflow construct represented by x (in case x represents a data variable definition) or a parameter of the workflow construct represented by x. Analogously, the sink of the data flow definitions is either the workflow construct represented by y or a parameter of the workflow construct represented by y.

The following data flows can be defined by a compound workflow definition x (the numbers indicate by which arrow the data flow definitions are represented in Figure 13):

- from an in or inout parameter of x to
    - (1) an in or inout parameter of a subworkflow of x or
    - (2) a data variable defined by x
- from a subworkflow sub_1 of x to
    - (3) an in or inout parameter of another subworkflow sub_2 of x or
    - (4) an out or inout parameter of x or
    - (5) a data variable defined by x

- from a data variable defined by x to
    - (6) an in or inout parameter of a subworkflow of x or
    - (7) an out or inout parameter of x

A data flow definition may optionally include a *data conversion function* which defines a mapping of data from the data format of the data source to that of the data sink.

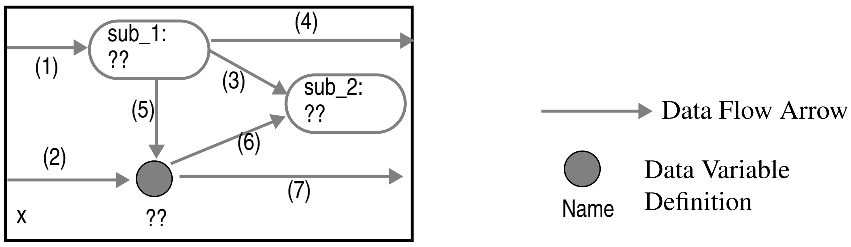

Figure 13: Data Flows Definable by a Compound Workflow x

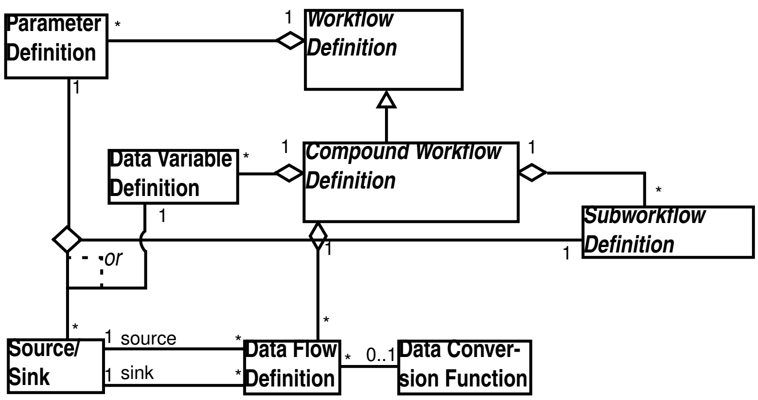

Figure 14: The TRAMs Core Workflow Model Classes of the Informational Perspective

The classes relevant in the informational perspective (shown in Figure 14) are defined as follows:

Definition 8 Workflow Definition (abstract class) (cont.)

Aggregations:

- parameter definitions: a set of objects of class Parameter Definition

Definition 17 Parameter Definition

Attributes:

- parameter definition in ODL: parameters are defined similarly to arguments of ODL-operations.

Associations:

- workflow definition the parameter is defined for: reference to an object of class Workflow Definition

Definition 10 Compound Workflow Definition (abstract class, subclass of Workflow Definition) (cont.)

Aggregations:

- data variable definition: a set of objects of class Data Variable Definition
- data flow definition: a set of objects of class Data Flow Definition

Definition 18 Data Variable Definition

Attributes:

- variable name and type definition (in ODL)

Definition 19 Data Flow Definition

Associations:

- source:
    - reference to one object of class Data Variable Definition or
    - reference to one object of class Parameter Definition and an optional reference to one object of class Subworkflow Definition
- sink:
    - reference to one object of class Data Variable Definition or
    - reference to one object of class Parameter Definition and an optional reference to one object of class Subworkflow Definition
- data conversion function that can translate data from the data format of the sink to that of the source: reference to an object of class Data Conversion Function
- compound workflow definition the data flow definition is defined for: reference to an object of class Compound Workflow Definition

Definition 20 Data Conversion Function

Attributes:

- textual description of the function

- definition of a function that converts data from one format to another

The Informational Perspective of the HIC Workflow Type

Table 1 shows the parameter definitions. In Table 2 the data flow definitions are listed and Figure 15 graphically represents them using data flow arrows.

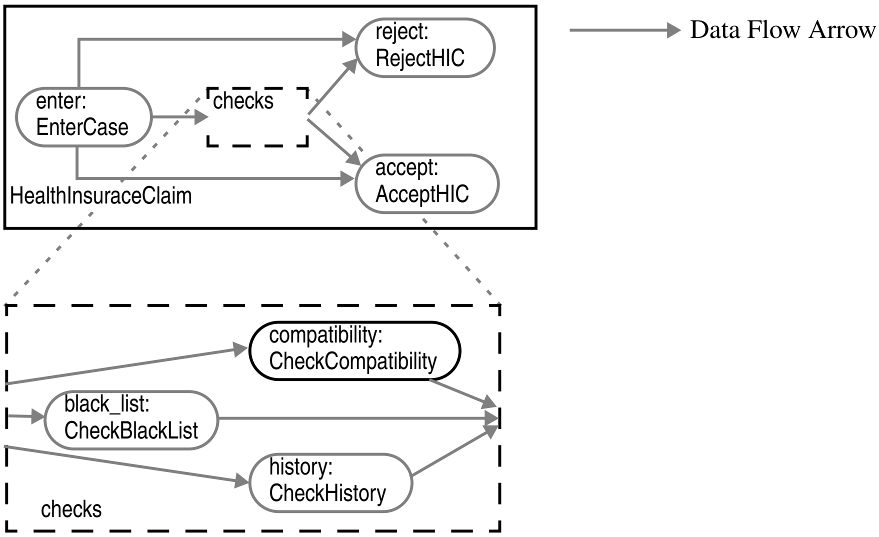

Figure 15: Informational Diagram: Health Insurance Claim Processing

Workflow Construct	Parameter Definitions
HealthInsuranceClaim	-
EnterCase	out policy_reference: PolicyReference, out cost: Cost, out treatment: Treatment, out diagnosis: Diagnosis
checks	in policy_reference: PolicyReference, in cost: Cost, in treatment: Treatment, in diagnosis: Diagnosis, out ok: Boolean, out reject_reason: RejectReason
CheckBlackList	in treatment: Treatment, out ok: Boolean, out reject_reason: RejectReason
CheckCompatibility	in treatment: Treatment, in diagnosis: Diagnosis, out ok: Boolean, out reject_reason: RejectReason
CheckHistory	in treatment: Treatment, in diagnosis: Diagnosis, in policy_reference: PolicyReference, out ok: Boolean, out reject_reason: RejectReason
RejectHIC	in policy_reference: PolicyReference, in cost: Cost, in treatment: Treatment, in diagnosis: Diagnosis, in reject_reason: RejectReason

Table 1: Parameter Definitions

Workflow Construct	Parameter Definitions
AcceptHIC	in policy_reference: PolicyReference, in cost: Cost,in treatment: Treatment, in diagnosis: Diagnosis

Table 1: Parameter Definitions

Workflow Construct Containing Data Flow Definitions	Source	Sink
HealthInsuranceClaim	enter.policy_reference enter.cost enter.treatment enter.diagnosis enter.policy_reference enter.cost enter.treatment enter.diagnosis enter.rejection_reason enter.policy_reference enter.cost enter.treatment enter.diagnosis	checks.policy_reference checks.cost checks.treatment checks.diagnosis reject.policy_reference reject.cost reject.treatment reject.diagnosis reject.rejection_reason accept.policy_reference accept.cost accept.treatment accept.diagnosis
checks	checks.treatment checks.treatment checks.diagnosis checks.treatment checks.diagnosis checks.policy_reference compatibility.ok history.ok black_list.ok compatibility.rejection_reason history.rejection_reason black_list.rejection_reason	black_list.treatment compatibility.treatment compatibility.diagnosis history.treatment history.diagnosis history.policy_reference checks.ok checks.ok checks.ok checks.rejection_reason checks.rejection_reason checks.rejection_reason

Table 2: Data Flow Definitions

4.2.3 Behavioral Perspective

In the behavioral perspective, the focus is on *when* workflows have to be executed. Since we assume processing entities being autonomous in the sense defined in Section 3.1, activities (that are carried out by these processing entities) are regarded as black boxes whose internal structures are not visible. This means that intermediate execution states of activities are not visible for the WFMS and that the WFMS can only control the start of activities.

In TRAMs, workflows run through the execution states *Initiated*, *Executing* and *Completed* during their lifetime (shown in Figure 16).

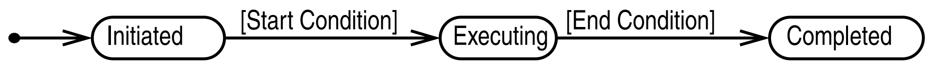

Figure 16: State Diagram of a Workflow in TRAMs (in Unified Notation[RSC97])

Execution states represent intervals of time and thus have a duration. State transitions on the other hand have no duration and happen instantaneously. Execution states are considered in more detail in Section 4.3.

When workflows have to be executed is specified by defining guards on the state transitions of workflows. A guard is a three-valued logical expressions that can be evaluated to either true, false or unknown. An elementary subexpression of a guard which does not include any of the logical operators $\neg$ (NOT), $\wedge$ (AND) and $\vee$ (OR) is evaluated to unknown, if it involves one or more references to in or inout parameters of workflows that are not in state Completed. Otherwise the subexpression is evaluated to either true or false. To evaluate expressions containing logical operators the usual rules of three-valued logic are used: $\neg$unknown = unknown, unknown $\wedge$ true = unknown, unknown $\wedge$ false = false, unknown $\vee$ false = unknown, unknown $\vee$ true = true.

A workflow may only change its execution state from state x to state y, if either no guard is associated with the transition connecting x to y or if the guard evaluates to true. For a workflow, two guards can be defined. The guard on the transition of a workflow leading from state Initiated to state Executing is called *start condition* and the guard on the transition that leads from Executing to Completed is called *end condition*.

Figure 17 below shows, that workflow definitions and workflow variable definitions may include the definition of a start condition and an end condition.

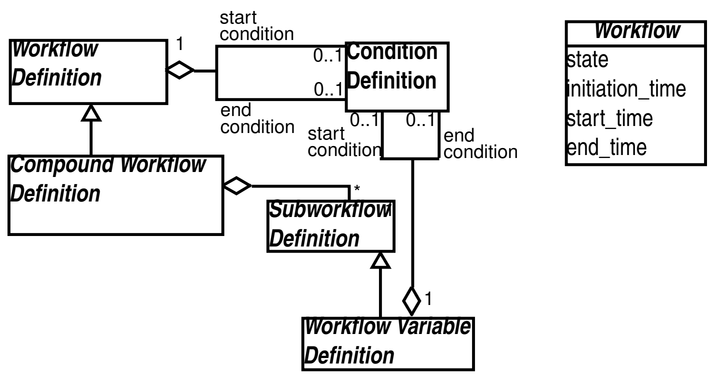

Figure 17: The TRAMs Core Workflow Model Classes of the Behavioral Perspective

In what follows we define in detail for each construct that can define start and end conditions to what these conditions can refer to.

Workflow Definition x

start and end condition:

- external variables that are not modified by any workflow end condition:

- in and inout parameters of x
- the following attributes of the workflow x_inst correspond to x:
    - initiation time: point in time in which x_inst has been created
    - start time: point in time when x_inst changes its execution state from Initiated to Executing

Compound Workflow Definition x (Complex Workflow Type and Block Definition)

end condition:

- data variables defined by x
- attributes of subworkflows of x:
    - execution states of the subworkflows
    - initiation time of subworkflows of x
    - start time of subworkflows of x
    - end time of subworkflows of x. The end time of a subworkflow is the point in time when the subworkflow changes its execution state from Executing to Completed.
- out and inout parameters of subworkflows of x

Subworkflow Definition x (Activity Type Variable, Complex Workflow Type Variable and Block Definition)

start and end condition:

- in and inout parameters defined by the compound workflow definition of which x is part of
- data variables defined by the compound workflow definition of which x if part of
- attributes of the compound workflow definition of which x is part of
    - initiation time
    - start time
- out and inout parameters of sibling workflows of x (they are defined as subworkflows by the compound workflow definition of which x is part of)
- attributes of sibling workflows of x:
    - execution state
    - initiation time
    - start time
    - end time
- external variables that are not modified by any workflow

The classes relevant in the behavioral perspective (shown in Figure 17) are defined as follows:

Definition 8 Workflow Definition (abstract class) (cont.)

Aggregates:

- start condition: an object of class Condition Definition
- end condition: an object of class Condition Definition

Definition 14 Workflow Variable Definition (cont.)

Aggregates:

- start condition: an object of class Condition Definition
- end condition: an object of class Condition Definition

Definition 21 Condition Definition

Attributes:

- a three-valued logical expression

Definition 22 Workflow (abstract class)

Attributes:

- execution state, where execution state ∈ {Initiated, Executing, Completed}
- initiation time: point in time in which a workflow has been created
- start time: point in time when a workflow changes its execution state from Initiated to Executing
- end time: point in time when a workflow changes its execution state from Executing to Completed

Graphical Representation of Start Conditions and End Conditions

Condition arrows allow to graphically represent the start and end conditions associated with workflow constructs (a workflow construct is either a complex workflow type, complex workflow variable, activity type, activity variable, block definition or external variable). There are two kinds of condition arrows: start condition arrows and end condition arrows. A *start condition arrow*, represented by a solid black arrow, connecting a construct x to a construct y means that the start condition of construct y involves one or more properties of construct x. Analogous, an *end condition arrow*, represented by a dotted black arrow, connecting a construct x to a construct y means that the end condition of construct y involves one or more properties of construct x.

The Behavioral Perspective of the HIC Workflow Type

Start and end conditions are listed in Table 3 and graphically represented in Figure 18.

Workflow Construct	Start Condition	End Condition
HealthInsuranceClaim	-	reject.state = Completed OR accept.state = Completed
enter	-	-
checks	enter.policy_reference != nil AND enter.cost > 500	(black_list.ok = false OR history.ok = false OR compatibility.ok = false) OR (black_list.ok = true AND history.ok = true AND compatibility.ok = true)
black_list	-	-

Table 3: Start and End Condition Definitions

Workflow Construct	Start Condition	End Condition
compatibility	black_list.ok = true	-
history	black_list.ok = true	-
reject	enter.policy_reference = nil OR enter.cost< 150 OR checks. ok = false	-
accept	enter.policy_reference != nil AND reject.state = Initiated AND (enter.cost<= 500 OR checks.ok =true OR (current_time -HealthInsurance- Claim.initiation_time) > 6 weeks))	-

Table 3: Start and End Condition Definitions

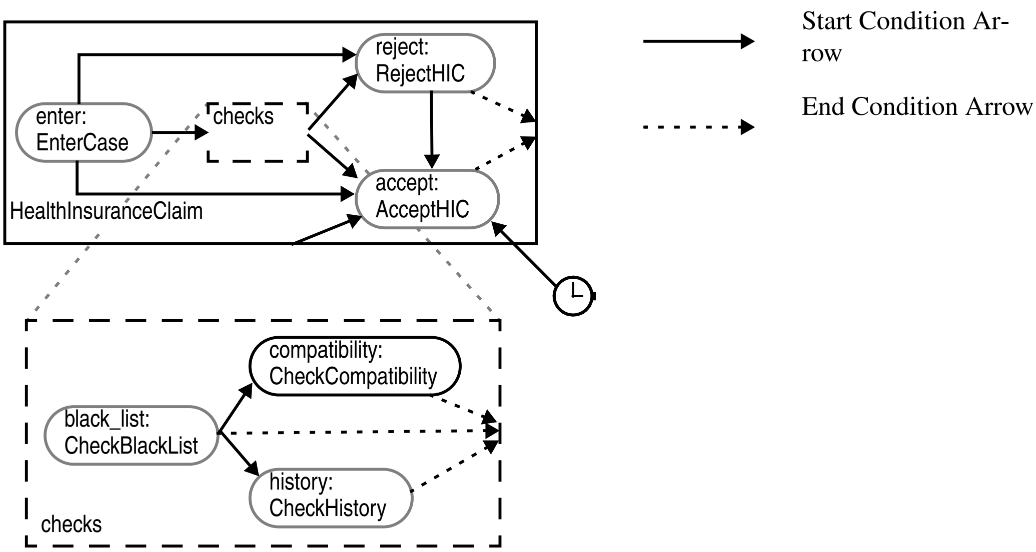

Figure 18: Behavioral Diagram: Health Insurance Claim Processing

4.2.4 Organizational Perspective

In the organizational perspective, the focus is on which processing entities carry out the activities workflows consist of. Before we show how the assignment of activities to processing entities can be specified, we introduce two associations between already defined classes (illustrated in Figure 19).

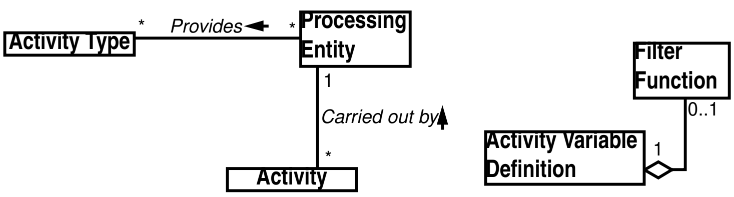

Figure 19: The TRAMs Core Workflow Model Classes of the Organizational Perspective

Processing entities provide activity types (represented by the association *"Provides*"). Furthermore, there exists an association "*Carried out by"* between activities and processing entities.

The specification of which processing entities should carry out which activities is done by *filter functions* defined by activity variables. A filter function defined by an activity variable v maps the processing entities that provide the activity type of v to a subset of these processing entities, thus it filters out the desired processing entities. The filter function is defined in OQL and may involve references to the same constructs as a start or end condition (as defined in Section 4.2.3) and additionally references to:

- attributes of processing entities:
  - the name of a processing entities: e.g. "activity x has to be carried out by the processing entity xy"
- relationships that exist among organizational entities
  - e.g. "activity x has to be carried out by a processing entity that belongs to the organizational unit xyz"
- a role: e.g. "activity x has to be carried out by a processing entity that can play the role y"
- previous assignments of activities to processing entities: "activity x must executed by the processing entity that carried out activity z"

The classes relevant in the informational perspective (shown in Figure 19) are defined as follows:

Definition 5 Processing Entity (Instance of class Organizational Entity Type)(cont.)

Associations:

- activity types that are provided by the processing entity: references to a set of objects of class Activity Type

Definition 23 Activity (subclass of Workflow)

Associations:

- assigned processing entity: reference to an object of class Processing Entity

Definition 15 Activity Variable Definition (subclass of Workflow Variable Definition) cont.)

Aggregations:

- filter function: an object of class Filter Function

Definition 24 Filter Function

Attribute:

- expression in OQL that returns a set of processing entities

Definition 9 Activity Type (subclass of Workflow Definition) (cont.)

Associations:

- processing entities that provide this activity type: references to a set of objects of class Processing Entity

The Organizational Perspective of the HIC Workflow Type

The HIC Workflow Type includes the definition of five activity variables. For each of them, a filter function is defined in Table 4. The filter functions involve references to processing entities, roles, and organizational units which are defined in Figure 6 in Section 4.1.

Activity Variable	Filter Function
enter	filters out the processing entities that can play the role "Clerk"
black_list	filters out the processing entities named "blacklist_checker"
history	filters out a set of processing entities that can play the role "Medical Specialist" and that belong to the same group as the processing entity that carried out the enter acitivity.
reject	filters out the processing entities named "customer information system"
accept	filters out the processing entities named "customer information system"

Table 4: Filter Functions defined in the HIC Workflow Type

4.3 Execution Model

The execution model specifies how workflows are executed. In order to precisely define the semantics of workflow executions, the workflow execution states introduced in Section 4.2.3, are further refined as shown in Figure 20.

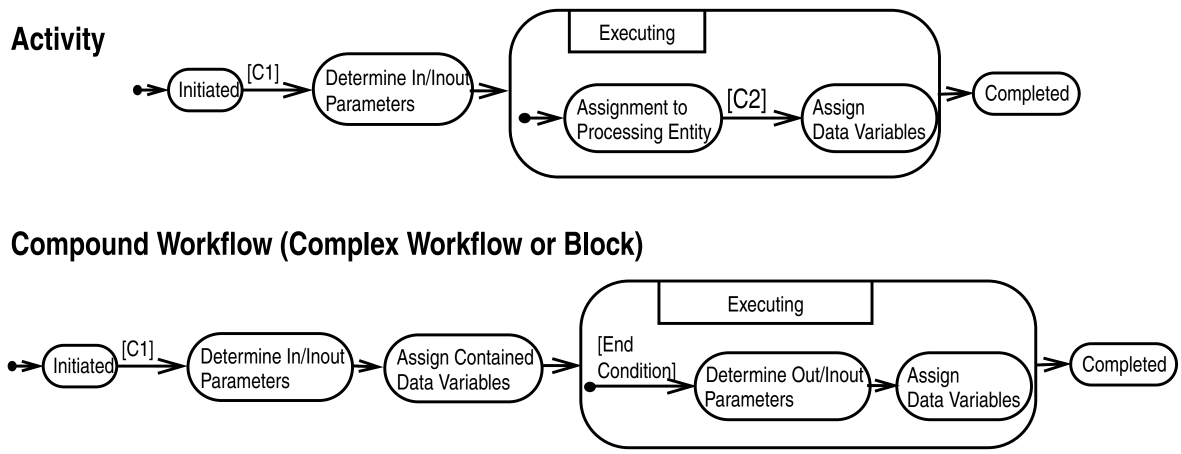

Figure 20: (UML) State Diagrams of Workflows

Generally, a workflow that is currently in execution state x can enter state y, if

- there is a transition defined leading from x to y and
- the steps to be carried out in x are done and
- in case there is a guard-condition defined for the transition connecting x to y: this condition evaluates to true

In what follows, we define for a workflow x what is done (the steps that are carried out) after x enters the execution states shown in Figure 20:

Initiated:

If x is a compound workflow:

- its subworkflows are created and their execution states are set to Initiated
- its contained data variables are created and initialized

Determine In/Inout Parameters:

If x is a subworkflow, the following steps are carried out:

1. The set of data flow definitions D defined by the compound workflow definition corresponding to the superworkflow of x is determined.
2. From D, those data flow definitions whose sink is an in or inout parameter of x are filtered out, resulting in a set D_1, which is a subset of D.
3. From D_1 those data flow definitions whose source is an out or inout parameter of a sibling workflow of x that is not in state Completed are excluded, resulting in a set D_2.
4. The data definitions D_2 are enforced. This means that for each data flow definition contained in D_1, which specifies a data flow from a source to a sink, the current value of the source is directly assigned to the sink, if there is no conversion function defined. Otherwise, the conversion function is applied to the current value of the sink and the result is assigned to the source.

If x is not a subworkflow, nothing needs to be done in this state.

Assign Contained Data Variables:

1. The set of data flow definitions D defined by the compound workflow definition corresponding to x is determined.
2. From D, those data flow definitions whose source is an in or inout parameter of x and (additionally) whose sink is a data variable contained in x are filtered out, resulting in a set D_1.
3. The data flow definitions contained in D_1 are enforced (as defined in step three in the above paragraph about the execution state Determine In/Inout Parameters).

Executing: -

Assignment to Processing Entity:

Upon entering the state Executing, an activity x of type X simultaneously enters the substate Assignment to Processing Entity. In this state, x is assigned to a processing entity according to the following steps:

1. The set P of processing entities that provide the activity type X is determined.
2. If x is a subworkflow and if the corresponding activity variable definition of x defines a filter function, it is applied on P which results in a new set P1. Otherwise P1 is equal to P.
3. If P1 consists of more than one processing entity, one processing entity is selected randomly from P1.

Determine Out/Inout Parameters:

1. The set of data flow definitions D defined by the compound workflow definition corresponding to x is determined.
2. From D, those data flow definitions whose sink is an out or inout parameter of x are filtered out, resulting in a subset D_1 of D.
3. The data flow definitions contained in D_1 are enforced.

Assign Data Variables

In state Assign Data Variables the following steps are carried out, if x is a subworkflow:

1. The set of data flow definitions D defined by the superworkflow of x is determined.
2. From D, those data flow definitions whose source is an out or inout parameter of x and (additionally) whose sink is a data variable contained in the superworkflow of x are filtered out, resulting in a set D_1.
3. The data flow definitions contained in D_1 are enforced.

If x has no superworkflow, nothing is done in state Assign Data Variables.

Completed: -

Next, the guard-conditions used in Figure 20 are defined. X denotes the respective workflow the state diagram refers to:

[C1]

The guard-condition [C1] evaluates to true if

- in case x is a subworkflow: its superworkflow is in state Executing and the start condition defined in the workflow variable definition corresponding to x evaluates to true and
- the start condition defined in the corresponding workflow definition evaluates to true

[C2]

The guard-condition [C2] evaluates to true if

- the processing entity assigned to x signals the end of x and
- if x has a superworkflow: the end condition defined in the activity variable definition corresponding to x evaluates to true and
- the end condition defined by the activity type of x evaluates to true

[End Condition]

The guard-condition [End Condition] evaluates to true if

- in case x has a superworkflow: the end condition defined in the subworkflow definition corresponding to x evaluates to true and
- the end condition defined in the workflow definition corresponding to x evaluates to true

4.4 Iteration and Parallel Blocks

With the constructs introduced so far, each workflow has to be specified individually. In particular it is not possible to specify a variable number (whose actual value will not be determined before execution time) of workflows of a certain type that have to be executed sequentially or that may be executed in parallel. In order to address these requirements, two new constructs, *Iteration Block Definition* and *Parallel Block Definition*, are introduced. As shown in Figure 21, iteration block definitions and parallel block definitions are specializations of block definitions.

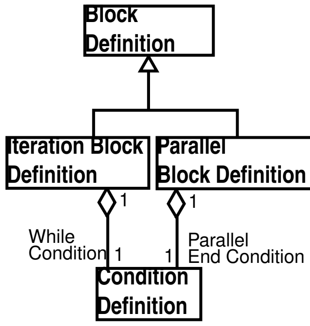

Figure 21: Iteration and Parallel Block Definition

A (normal) block definition, an iteration block definition and a parallel block definition all include the specification of a set of subworkflows (blocks and workflow variables) and a set of data variables. If a block is created (as part of a compound workflow) exactly one version of each subworkflow and data variable specified by its according block definition is created. On the other hand, after an iteration block has been created, a new set of the subworkflows is created one after another, as long as certain conditions defined in the according iteration block definition hold. Finally, after a parallel block has been created, the subworkflows defined are created n-times, depending on the number of in and inout parameter versions that have been assigned to the parallel block.

4.4.1 Iteration Block Definition

In addition to the information contained in a "normal" block definition, an iteration block definition includes a *while condition* which determines at runtime how many times a new set of the subworkflows (called *iteration set*) defined by the iteration block has to be created and executed one after another. The data variables defined by an iteration block are created only once while the iteration block is in the execution state Initiated. Thus, data flows among subworkflows belonging to different iteration sets can be defined using references to data variables.

A while condition wc can involve references to the same constructs as a start or end condition. A reference in wc to an attribute or a parameter of a subworkflow contained in the iteration block for which wc is defined always relates to the subworkflow contained in the iteration set that was created last.

Figure 22 presents the state diagram of iteration blocks. The execution state Initiated differs from that of a compound workflow (as defined in Section 4.3) in that only the defined data variables are created but not the subworkflows. The subworkflows are created in the execution state Create Subworkflows.

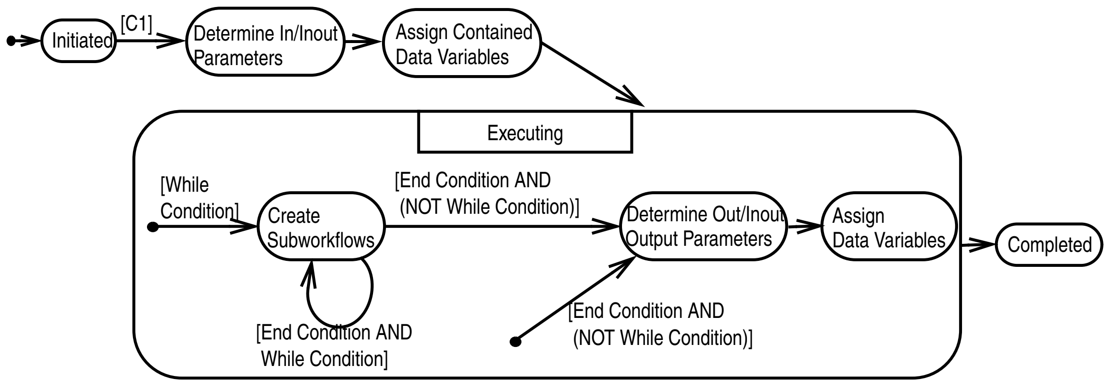

Figure 22: State Diagram of Iteration Blocks

The Iteration Block Definition class is defined as follows:

Definition 25 Iteration Block Definition (subclass of Block Definition)

Aggregations:

- while condition: an object of class Condition Definition

Figure 23 illustrates a simplified complex workflow type CreateAndReviewDesign involving the creation and reviewing of a design document. Table 5 through Table 8 define the parameters, data variable, data flows, and conditions. As a first step, a design document is created (create). Next, the document is reviewed (review). After the review step, the workflow terminates, if the review step accepts the document (the out parameter ok of the review step has the value TRUE). Otherwise, the document needs to be modified (modify) and reviewed again. The modify/review cycle is repeated until the review step accepts the design document, i.e. its out parameter has the value TRUE.

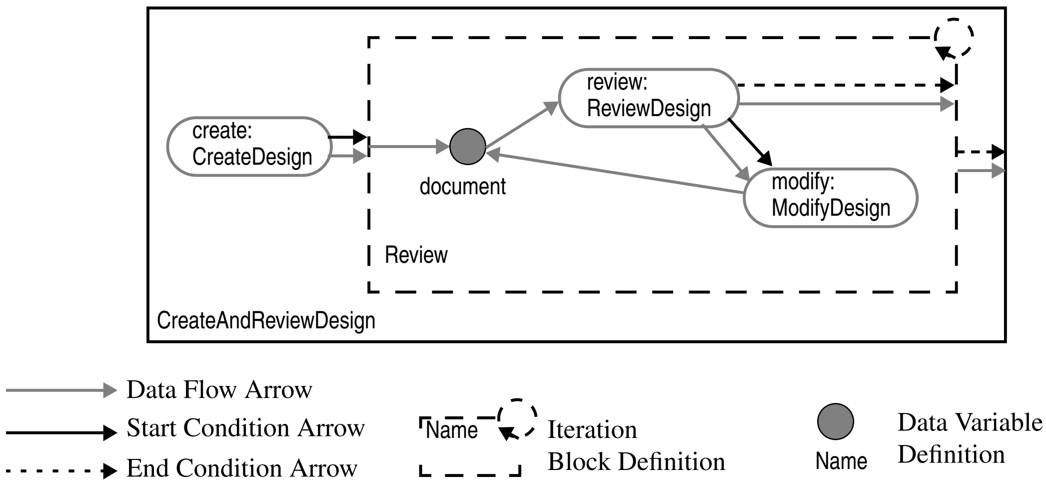

Figure 23: Complex Workflow Type CreateAndReviewDesign

Workflow Construct	Parameter Definitions
CreateAndReviewDesign	out design_document: DesignDocument
CreateDesign	out design_document: DesignDocument
Review	inout design_document:DesignDocument
ReviewDesign	inout design_document:DesignDocument , out ok: Boolean
ModifyDesign	inout design_document DesignDocument

Table 5: Parameter Definitions

Defining Workflow Construct	Data Variable Definition
CreateAndReviewDesign	DesignDocument document

Table 6: Data Variable Definitions

Workflow Construct Defining Data Flows	Source	Sink
CreateAndReviewDesign	create.design_document Review.design_document	Review.design_document CreateAndReviewDe-sign.design_document
Review	Review.design_document document review.design_document review.design_document modify.design_document	document review.design_document Review.design_document modify.design_document document

Table 7: Data Flow Defintions

Workflow Construct Defining Conditions	Start Condition	End Condition	While Condition
CreateAndReviewDe-sign	-	Review.state = completed	-
Review	create.state = completed	modify.state = completed OR review.ok = TRUE	review.ok = FALSE
modify	review.ok = FALSE	-	-

Table 8: Condition Definitions

4.4.2 Parallel Block Definition

A parallel block definition contains all the information of a “normal” block definition, but the information sometimes has a different meaning. A parallel block, as a normal block, defines a set of subworkflows. But in contrast a normal block, the subworkflows of a parallel block must be created and executed n-times in parallel. One set of subworkflows is called *parallel set*.

As a normal block definition, a parallel block definition includes a set of in, inout, and out parameter definitions. In contrast to normal blocks, these definitions do not refer to the parallel block itself but to its parallel sets. A parallel block has exactly one input and one output parameter. The input parameter, which is called `in_parameter`, is of (ODL) type set, whose elements are (ODL) structs consisting of the in and inout parameters defined by its parallel block definition. The output parameter, called `out_parameter`, is also of type set and its elements are structs consisting of the out and inout parameters defined by the parallel block definition. Assume a parallel block pbd1 definition includes the following parameter definitions:

```
in x:integer, inout y:integer, out name:string
```

Then, the actual parameters of a parallel block of kind pbd1 would be the followings:

```
in in_parameter: Set<struct<x: integer, y:integer>>,
out out_parameter: Set<struct<y:integer, name:string>>
```

Assume that a compound workflow definition x (complex workflow type, block definition, iteration block definition or parallel block definition) includes a parallel block definition pdb. Then, the `in_parameter` and the `out_parameter` of pdb are the only parameters of pdb to which data flow definitions of x can refer to.

At execution time, for each element in the `in_parameter` of a parallel block a parallel set is created and executed. Each of these parallel sets produces one element of the `out_parameter` of the parallel set. The *parallel end condition* defines when a parallel block can enter the state Completed.

State Diagram

Figure 24 shows the state diagram of parallel blocks.

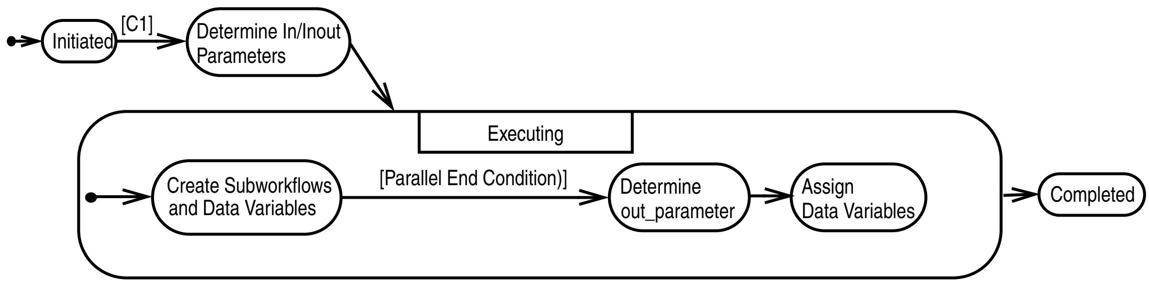

Figure 24: State Diagram of a Parallel Block

In what follows we consider the execution states from Figure 24. We define for a parallel block p what is done after p enters the one of the following execution states:

Initiated: -

Create Subworkflows and Data Variables:

For each element of the parameter `in_parameter` of p the following steps are carried out:

- the subworkflows (defined by the parallel block definition of p) are created and their execution states are set to Initiated.
- the data variables (defined by the parallel block definition of p) are created and initialized.

Thus, if `in_parameter` of p contains 20 elements when p enters the state Create Subworkflows and Data Variables the subworkflows and data variables are created 20 times. Each created set of subworkflows and data variables is called *parallel set*. Each parallel set is executed like a "normal" block as defined in Section 4.3.

Determine out_parameters:

The out_parameter of p is built from the out and inout parameters of the parallel sets contained in p.

Assign Data Variables:

1. The set of data flow definitions D defined by the compound workflow definition corresponding to the superworkflow of p is determined.
2. From D, those data flow definitions whose source is the out_parameter of p and (additionally) whose sink is a data variable contained in the superworkflow of p are filtered out, resulting in a set D_1.
3. The data flow definitions contained in D_1 are enforced.

Next, we consider the Parallel End Condition of parallel block definitions:

[Parallel End Condition]

A parallel end condition of a parallel block p can involve existential and universal quantifiers, e.g.:

- `for_all: state = completed`
  evaluates to true, if all parallel sets of p are in state completed
- `exists: state = completed`
  evaluates to true, if at least one parallel set of p is in state completed

The Parallel Block Definition class is defined as follows:

Definition 26 Parallel Block Definition (subclass of Block Definition)

Aggregations:

- parallel end condition: an object of class Condition Definition

Example

Figure 25 shows a workflow type specifying a conference paper collection and reviewing process. Table 9 to Table 11 define parameters, data flows, and conditions.
As a first step, papers are collected (collect_papers). After the conference deadline is reached, for each of the papers an own subworkflow is started. Each subworkflow consists of an assignment step(assign), which assigns a paper to a reviewer, followed by a collection step, which collects the review (collect_review).

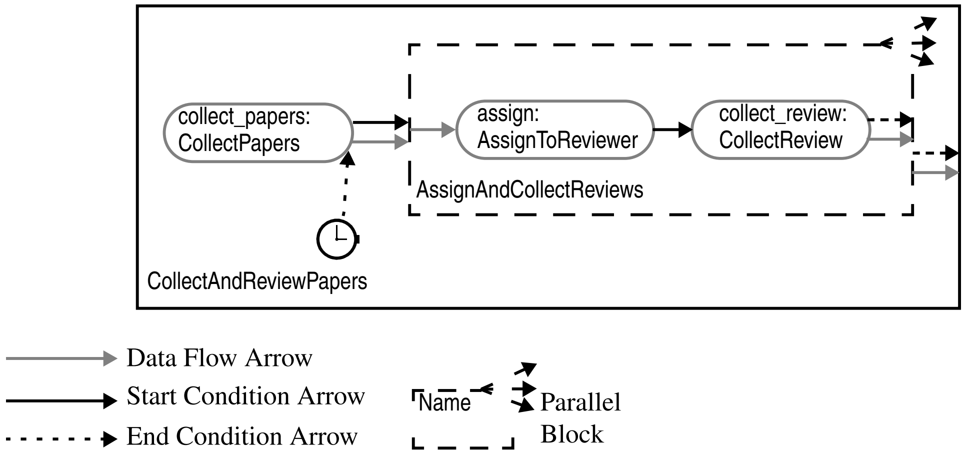

Figure 25: Conference Paper Collection and Review Workflow Type

Workflow Construct	**Parameter Definitions**
CollectAndReviewPapers	in deadline: Date, out reviews: Set<struct<review: Review>>
CollectPapers	out papers: Set<struct<paper:Paper>>
AssignAndCollectReviews	in paper: Paper, out review: Review
AssingToReviewer	in paper: Paper
CollectReview	out review: Review

Table 9: Parameter Definitions

Workflow Construct Containing Data Flow Definitions	**Source**	**Sink**
CollectAndReviewPapers	collect_papers.papers AssignAndCollec-tReviews.out_parameter	AssignAndCollec-tReviews.in_parameter CollectAndReviewPapers.reviews

Table 10: Data Flow Defintions

Workflow Construct Containing Data Flow Definitions	Source	Sink
AssignAndCollectReviews	AssignAndCollectReviews.paper collect_review.review	assign.paper AssignAndCollectReviews.review

Table 10: Data Flow Defintions

Workflow Construct	Start Condition	End Condition	Parallel End Condition
CollectAndReviewPapers	-	AssignAndCollectReviews.state = completed	-
collect_papers	-	(CollectAndReviewPapers.deadline - current_time > 0)	-
AssignAndCollectReviews	collect_papers.state = Completed	collect_review.state = completed	for_all: state = completed
collect_review	assign.state = Completed	-	-

Table 11: Condition Definitions

4.5 Time Issues

Time is an important issue in workflow management. In particular, a workflow model should support the specification of temporal constraints for workflows, including points in time when workflows have to be started or deadlines until which workflows have to be completed.

Basic Concepts of Time

There exist various models of time. It is common to distinguish between *continuous* and *discrete* time models. While in continues models, time is isomorphic to real numbers, in discrete models it is isomorphic to natural numbers. The following definitions are taken from [DGK+97]. An *instant* is a time point on an underlying time axis. The time between two instants is a *time interval*. An amount of time with known length, but with no specific starting or ending instants is a *span*. A span is either positive, denoting forward motion of time, or negative, denoting backward motion in time. A span can be *fixed* or *variable.* A fixed span is independent of the context, e.g. “one hour”. “One month” in the Gregorian calendar is an example of a variable span.

In discrete models, a *chronon*, which is a non-decomposable positive span, is assumed. For our approach we use a discrete model of time with a chronon of a minute. We assume that a global variable `current_time` exists that contains the actual date and time. Furthermore, several instants relating to workflows can be referred to, which are modeled as attributes of workflows (see Section 4.2.3):

- `x.initiation_time`: instant in which workflow x has been created
- `x.start_time`: instant in which workflow x changes its execution state from Initiated to Executing

- `x.end_time`: instant in which workflow x changes its execution state from Executing to Completed

The above three attributes of workflows either have an instant (e.g. 97.02.18,13:30) or "unknown" as value. Upon the creation of a workflow x, `x.initiation_time` is set to `current_time`, `x.start_time` and `x.end_time` are set to "unknown". At the time x changes its execution state from Initiated to Executing, `x.start_time` is set to `current_time`. Eventually, when x changes its state from Executing to Completed, `x.end_time` is set to the value of `current_time`.

Defining Instants

We support two kinds of instant definitions: *absolute instant definitions* and *relative instant definitions*:

- An absolute instant definition consists of a year, month, day, hour, and minute e.g. 97.02.18,13:15.
- A relative instant definition is of the form *(i+s)* or *(i-s)*, where *i* is an instant and *s* is a span. If *i* or (inclusive or) *s* has the value "unknown", then also *(i+s)* and *(i-s)* have the value "unknown".

Defining Spans

In order to prevent unclarities, only the definition of fixed spans is supported. They can be defined in one of the following ways:

- a number of years, weeks, days, hours and minutes, e. g., 0 years, 3 weeks, 3 days, 5 hours, 30 minutes.
- *(i1-i2)*, where *i1* and *i2* are instants (if *i1* or *i2* has the value "unknown", then *(i1-i2)* also has the value "unknown");
- *(s1+s2)* or *(s1-s2)*, where *s1* and *s2* are fixed spans;(if *s1* or *s2* has the value "unknown", then also *(s1+s2)* and *(s1-s2)* have the value "unknown");

Comparing Spans and Instants in Start and End Conditions

Start and end conditions of workflow definitions and subworkflow definitions can include comparisons of instants and spans. An instant can be compared to other instants and a span can be compared to other spans, using a binary comparison operator o ∈ {<, >, =, !=, <=, >=} which either maps two instants or two spans to either "true", "false" or "unknown". If one or both operands of an operator o has the value "unknown", then o maps them to "unknown". Otherwise o maps to either "true" or "false". For example the expression $s1 < s2$ maps to true, if span *s1* is shorter than span *s2*. It maps to "false", if *s1* is longer or as long as *s2.* Another example is the start condition of the subworkflow definition accept of the HIC example which consists of the following subexpression: (current_time -HealthInsuranceClaim.initiation_time) > 6 weeks.

Defining Deadlines for Workflows

We support the specification of deadlines for workflows. A *deadline* associated with a workflow x is an instant until which x should be in state Completed. Deadlines can be defined by workflow definitions and workflow variable definitions. A deadline specification is an instant definition that either evaluates to a point in time or to unknown. Following we define for each construct to what a deadline specification defined by it refer to:

Workflow Definition x

- global variable `current_time`
- in or inout parameter of x
- the following attributes of x:
    - initiation time
    - start time

Compound Workflow Definition x

- out and inout parameters of subworkflows of x
- attributes of subworkflows of x:
    - initiation time
    - start time
    - end time
    - deadline

Subworkflow Definition x

- global variable `current_time`
- out and inout parameters of the compound workflow definition x is part of
- attributes of the compound workflow definition x is part of:
    - initiation time
    - start time
- out and inout parameters of sibling workflows of x
- attributes of sibling workflows of x
    - initiation time
    - start time
    - end time
    - deadline

It is important to note that the deadline specification of a subworkflow definition x overrides the deadline specification defined by the workflow type of x.

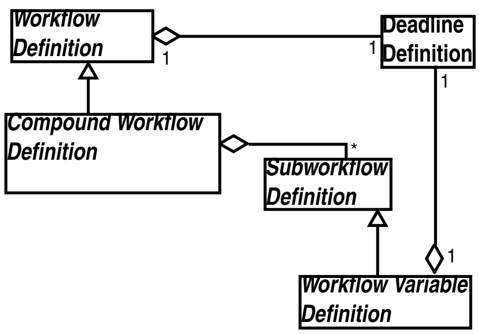

Figure 26: Classes Workflow Definition and Workflow Variable Definition Extended with Deadline Attribute

The classes Workflow Definition and Workflow Variable Definition are extended as follows:

Definition 8 Workflow Definition (abstract class) (cont.)

Aggregation:

- deadline: an object of class Deadline Definition

Definition 14 Workflow Variable Definition (cont.)

Aggregation:

- deadline: an object of class Deadline Definition

Definition 27 Deadline Definition

Attribute:

- deadline definition: specification of an instant that either evaluates to a point in time or to "unknown"

4.6 Inheritance Relationships between Complex Workflow Types

In analogy to inheritance relationships between classes in object-oriented data models, we introduce the notion of *inheritance relationships* between complex workflow types (CWTs).

An inheritance relationship associates two CWTs, where one CWT called *subtype* inherits the features of the other one called *supertype*. More precisely, the features of the supertype comprise the subworkflow definitions, data variable definitions, start and end conditions, deadline definition, and data flow definitions. Additionally the subtype may add to or modify the features it inherits in ways that will be described below. In order to keep the approach simple, only single inheritance is supported, i.e. a complex workflow type can have an arbitrary number of subtypes, but at most one supertype. Thus, the inheritance relationships among CWTs result in *inheritance hierarchies* which are trees with CWTs as nodes and inheritance relationships as edges.

A subtype can add to and modify the features it inherits from its supertypes in the following ways:

- add workflow constructs:
  - new subworkflow definitions directly to the supertype and to block definitions of the supertype
  - new data flow definitions directly to the supertype and to block definitions of the supertype
  - new parameter definitions to those of its supertype and those of the block definitions of the supertype
- overwrite workflow constructs:
  - overwrite the start and end condition definitions of its supertype and those specified by the subworkflow definitions of the supertype
  - overwrite the while condition of iteration block definitions of the supertype
  - overwrite the parallel end condition of iteration block definitions of the supertype
  - overwrite the filter functions of the activity variable definitions of the supertype
  - overwrite the deadlines defined by the supertype and those of the subworkflow definitions of the supertype

CWTs are specified as *abstract* or *concrete*. While of a concrete CWT any number instances can be created, no instances can be created of an abstract CWT.

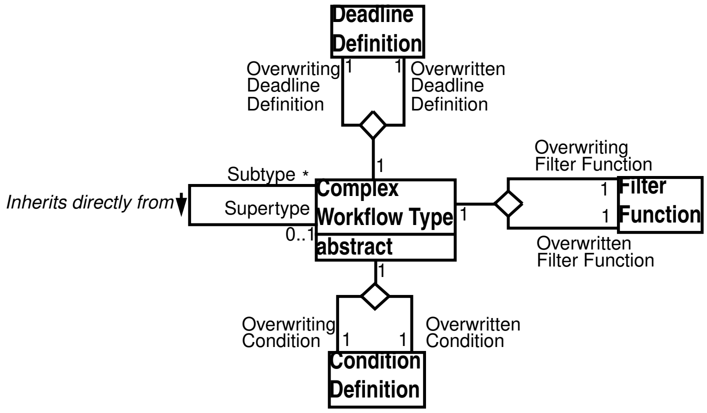

Figure 27: Additional Associations Between Already Defined Classes to Support Inheritance Relationships Among Complex Workflow Types

The classes of Figure 27 are defined in the following way:

Definition 11 Complex Workflow Type (subclass of Compound Workflow Definition) (cont.)

Attributes:

- indication if the complex workflow type is abstract or not: boolean

Associations:

- supertype: reference to an object of class Complex Workflow Type
- deadline definitions that overwrite existing ones: references to a set of objects of class Deadline Definition
- filter functions that overwrite existing ones: references to a set of objects of class Filter Function
- condition definitions that overwrite existing ones: references to a set of objects of class Condition Definition

Definition 19 Data Flow Definition (cont.)

Associations:

- overwritten data flow definition: optional reference to an object of class Data Flow Definition

Definition 21 Condition Definition (cont.)

Associations:

- overwritten condition definition: optional reference to an object of class Condition Definition

Definition 24 Filter Function (cont.)

Associations:

- overwritten filter function: optional reference to an object of class Filter Function

Definition 27 Deadline Definition (cont.)

Associations:

- overwritten deadline definition: optional reference to an object of class Deadline Definition

5 References

[Cat94] R. G. G. Cattell. *Object Data Management: Object-Oriented and Extended Relational Database Systems*. Addison-Wesley, 1994.

[CKO92] B. Curtis, M.I. Kellner, and J. Over. Process Modelling. *Communications of the ACM*, 35(9):75–90, September 1992.

[DGK+97] C. Dryeson, F. Grandi, W. Kaefer, N. Kline, and N. Lorentzos et al. A Consensus Glossary of Temporal Database Concepts. Technical report, http://www.cs.auc.dk/ csj/Glossary/, 1997.

[GKT95] A. Geppert, M. Kradolfer, and D. Tombros. Realization of Cooperative Agents Using an Active Object-Oriented Database Management System. In T. Sellis, editor, *2nd Intl. Workshop on Rules in Database Systems (RIDS)*, Athens, Greece, September 1995. Lecture Notes in Computer Science, Springer.

[LA94] F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Systems Journal*, 33(2):326–348, 1994.