Activity Nets:
A UML profile for modeling workflow and business processes

Author: Gregor v. Bochmann, SITE, University of Ottawa

(August 27, 2000)

1. Introduction

1.1. Purpose of this document

Workflow modeling is important in a large range of application domains, including business process architectures, factory automation, and various work processes, such as software engineering processes. Workflow modeling is characterized by a high level of abstraction and the concentration on such concepts as work activities, resources required to perform these activities, and the objects that are created and transformed by these processes. In order to obtain useful models one needs a suitable notation for describing workflow models, and automated tools for verifying the consistency of these models and analyzing their performance. This document describes such a notation, which is similar to UML Activity Diagrams [1].

This document defines a UML profile, called Activity Nets, which is closely related to Activity Diagrams. The profile includes modeling concepts that have proven suitable for modeling business process architectures and other activies of concurrent processes. The basic concepts are *activities*, and *participants* that participate in the activities, such as actors, resources and created or consumed objects. These concepts are part of the OPAL modeling language [5]. They are very similar to the UML concepts *ActionState* and *ObjectFlowState*, respectively, which are used in the context of UML Activity Diagrams. These concepts are also related to the new semantics of actions described in [2]. The here defined profile provides a precise dynamic semantics for these modeling concepts which can be used as a basis for the construction of automated analysis tools which provide performance simulations for the established models (for an example of such a tool, see [5]).

It is noted that the semantics of Activity Nets is very similar to the one of UML Activity Graphs, but defined independently of State Machines. Therefore this semantic definition could be used, after small adaptations, as an alternative to the semantic definition of Activity Diagrams based on state-machine oriented concepts as presently defined in UML [1].

1.2. Overview

In this section we provide a cursory high-level overview of the basic concepts of the profile and the motivation behind them. See references [5, 6, 7] for more comprehensive descriptions.

1.2.1. Semantics

As shown in Figure 1, an *ActivityNet* consists, at the semantic level, of a collection of *Activities* and *ParticipantSets*, the latter representing sets of actors or resources. The ParticipantSets are related to the Activities by *use*, *change* or *do* relations.

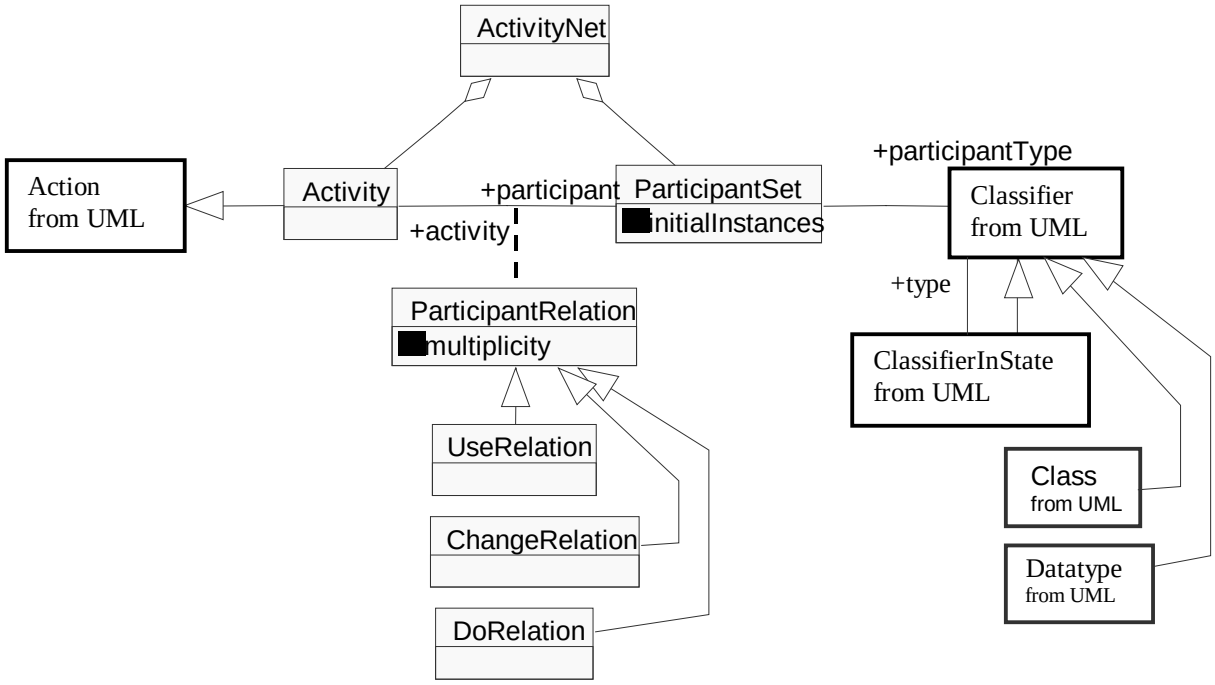

Figure 1: Semantic concepts of Activity Nets

A UseRelation between a ParticipantSet and an Activity means that one or several instances of the participants of the set (as indicated by the multiplicity attribute of the ParticipantRelation) are required for the execution of the activity and will be consumed or transformed. A ChangeRelation between a ParticipantSet and an Activity means that one or several instances of participants of that set will be created when the activity is executed, or will result from an use-related participant through a transformation. In the case of a transformation, the result of the transformation may be an instance of the participant in a different state. The participants of a ParticipantSet must be of a given type, as defined by a UML *Classifier* which is either a UML *Class*, a *DataType* or a *ClassifierInState*. The dynamic behavior of the participants may be defined in terms of a state diagram including the different states during the dynamic evolution of an object instance. If the type of the objects of an ParticipantSet is a ClassifierInState then these objects must be in a particular state.

A DoRelation between a ParticipantSet and an Activity means that one or several instances of the participants of the set are required during the execution of the activity, however, these participants are not consumed nor modified. Examples of such participants are actors that perform the activity, or non-consumable resources, e.g. CPU processing power or information from a database.

At the basic semantic level, an ActivityNet defines the order in which the different activities may be executed. The precondition for the execution of an activity is the presence of a sufficient number of participant instances in the ParticipantSets related by the use and do relations. When the activity starts its execution, these instances are removed from their ParticipantSets, and when the activity completes, the instances from do-related ParticipantSets are reconstituted and new instances are inserted in the change-related ParticipantSets. The initial number of instances in the ParticipantSets is indicated by the initialInstances attribute. We note that the basic semantics of Activity Nets is the same as Petri nets [10], a well-known formalism for modeling systems with concurrency. This is an advantage because techniques and tools developed for the analysis of Petri nets can be adapted for the analysis of Activity Nets.

A second level of semantics, called Stochastic Activity Nets, can be provided for performance simulations and is related to the semantics of stochastic Petri nets [11]. At this level, each activity is characterized by its execution time, which may either be a constant or a probability distribution, such as for instance a uniform distribution between a minimum and a maximum value. Models at this level allows for performance evaluation using analytical or simulation tools.

Another, complementary refinement of the semantics, in the following called Attributed Activity Nets, allows the consideration of attributes of participants and additional preconditions for activities depending on the attribute values of the object instances participating in the activity. At this level, it becomes important to consider the order in which the participants of a given ParticipantSet are created and consumed, because they are identified by their characteristic attribute values. Normally it is assumed that each ParticipantSet enforces a FIFO ordering.

1.2.2. Notation

The proposed notation for Activity Nets is closely related to the notation of UML Activity Diagrams. An activity is represented as a UML ActionState. A ParticipantSet is represented as a UML ObjectFlowState. The ParticipantRelations are represented as dashed lines (like the UML Action-Object Flow Relationships, see [1] Section 3.89) between the related Activity and ParticipantSet with arrows indicating the nature of the relation: A single arrow pointing to the Activity indicates a UseRelation. A single arrow pointing to the ParticipantSet indicates a ChangeRelation, and a double arrow indicates a DoRelation. An example ActivityNet is shown in Figure 2. The notation for the ParticipantRelations is simpler than the corresponding notation for UML Activity Diagrams since no *join* and *fork* states are required (see Figure 4).

Some additional notation is foreseen to indicate exclusive OR relationships between several change relations (see Section 3.2.3). Another abbreviated notation is introduced (see Sections 3.2.1 and 3.2.4) for representing directly the order of execution of different

activities (without the intermediate ParticipantSets). It is also proposed (see Section 3.2.2) that DoRelations may be represented in an alternate form, for instance using the Swimlane notation of the UML Activity Diagrams (see [1] Section 3.88) or the "component" notation proposed by Use Case Maps [9]. Notations for substructure of Activities are discussed in Section 3.2.5.

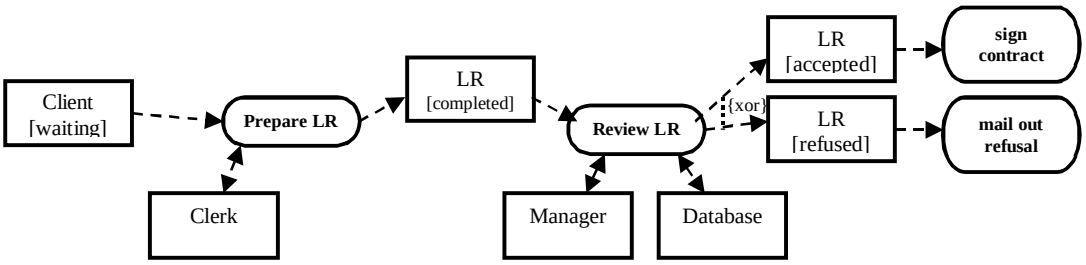

Figure 2: Example of an Activity Net: Processing of a loan request (LR)

1.2.3. Relationship with other UML views

Activity nets are typically of a more abstract nature than certain other dynamic views, such as UML Sequence and Collaboration diagrams. ActivityNet diagrams are therefore suitable for making early abstract models which closely relate to UML Use Cases. In contrast to use cases, however, they may include part of the internal structure of a system or organization which provides the services described by the use cases. Sequence and collaboration diagrams, together with more detailed UML Static Structure diagrams may be used during the later phases of the system development process for a more implementation-oriented description of the system.

Typically, an activity net is complemented with diagrams showing the classification hierarchy of the participants involved in the activities. Sometimes this information may be complemented with associations between the different classes of participants; these associations may represent containment relationships or transformation relationships related to the activities of the described processes. If the state transitions of the participants are important to be documented, UML StateChart diagrams may be used for this purpose. This is further discussed in Section 3.3.

1.2.4. Relation to other UML developments

As mentioned above, Activity Nets are conceptually quite close to Activity Diagrams, although their semantics is defined in a different manner. The concepts of Activity Nets are also relevant in the context of other ongoing UML developments, such as the following.

(a) Action Semantics

Work is ongoing to define the semantics of actions [2]. In this context, *data flow* relations are defined which are similar to the ParticipantRelations (*Use* and *Change*) of Activity

Nets. One could define data flow diagrams as a specialization of Activity Nets, as explained in more detail in Section 2.6.

(b) Revised semantics for Activity Diagrams

In the context of the development of UML Version 2, work is ongoing to revise the semantic definition of Activity Diagrams. Given the similarity of the modeling concepts, the semantics of Activity Nets may be used for this purpose.

(c) Event-Based Architectures in Enterprise Application Integration (EAI)

A new OMG request for proposal (RfP) on EAI has a due date of November 30, 2000. It appears that the concepts of Activity Nets are quite relevant in this context. For instance, an EAI *Business Model* , as defined in the RfP, could be modeled as an Activity Net. An EAI *Business Event* may be modeled by the arrival of an object in a ParticipantSet. In fact the *loose coupling* to be realized in EAI by messaging can be realized by the objects in ParticipantSets and their Use and Change relations with Activities. Concerning the *Data-Based Architecture* of EAI, it could be realized by the use of Do relationships which relate an Activity with a data resource (represented by the related ParticipantSet) which is used or updated by the Activity.

(d) Software Process Engineering (SPE)

The software engineering process may be considered as a special case of workflow. A OMG working group is in the process of considering the proposals that have been submitted in response to the OMG SPE Management Request for Proposals (RfP). For the software engineering process many specific types of activities, deliverables, resources and techniques must be defined, however, they may be considered specializations of the Activities and Participants of Activity Nets. If we consider in particular the Initial Submission from Fijitsu/DMR [13], we may note the following. The *Process Component* (see for instance Figure 3-3 of the Fijitsu/DMR submission) can be identified with an Activity (of Activity Nets). A *Process Component* may be refined and described by subprocesses, like an Activity may be defined with a substructure (as described in Section 3.2.5 below). In such a way, a *Phase* may consist of several subprocesses, each resulting in a particular *Milestone*, which can be modeled as a Participant created by the subprocess Activity (as indicated by a ChangeRelation). The *Resource Types* defined in Figure 3-4 of the submission may be considered to be specializations the UML Classifier (or Class) which defines the types of Participants in the Activity Net model of Figure 1. A *Deliverable Types* could normally be modeled by a ParticipantSet which has a ChangeRelation with the process that created the deliverable, and a UseRelation with the process that uses that deliverable. *Information Units* and *Teams* may be represented by ParticipantSets which have a DoRelation with the processes that use the information unit or with the team that performs the process. Therefore it seems that the concepts of Activity Nets correspond to the Metaclass definitions given in Section 3.2 of the submission. A corresponding relationship with UML Activity Diagrams is already indicated in Figures 3-3 and 3-4 of the submission.

(e) Workflow Management Facility

A document submitted to OMG by the Workflow Management Coalition [12] describes workflow interfaces which should provide the possibility for interworking between different workflows, possibly modeled with different tools. The core workflow interfaces enumerated in Section 2.1.1 of [12] have some similarity with the concepts of Activity Nets. In particular, a *WfActivity* may be identified with an Activity. Also a *WfProcess* and *WfRequester* may be modeled by an Activity. A *WfProcess* could typically be modeled by an activity which is refined into a number of subactivities, as described in Section 3.2.5 below. A *WfResource* could be modeled as a ParticipantSet which is related by a DoRelation with the activity that requires the resource.

(f) Enterprise Distributed Object Model (EDOC)

To some extend, Activity Nets may also be useful as concepts for the modeling of Enterprise Distributed Object Models as defined in the OMG Request for Proposals on EDOC. In fact, the Initial Submission from SUN contains the notion of execution order of different activities, as described in Section 3.2.1 of this document.

2. Semantics

2.1. Semantic overview and relation with standard UML

This profile is based on the UML standard and does not require any other prerequisite profiles.

This profile is additive; in other words, it can be used in conjunction with any other meta-classes from standard UML. However, for the purpose of modeling the dynamic aspects of business workflow architectures, it is suggested to only use a subset of the standardized UML notations, as discussed in Section 3.3.

The abstract syntax of the concepts introduced in this profile is given in Figure 1. The abstract syntax of the mapping of these concepts to the UML concepts of Activity Graphs is shown in Figure 3.

The basic semantics of Activity Nets is described first in a stand-alone fashion, independently of UML, in Section 2.2. The same semantics can be obtained through the mapping of Activity Nets to UML concepts as shown in Figure 3 and explained in Section 2.3. We note that this mapping is based on UML Activity Graphs. However, an alternate mapping could be based on the action semantics defined in [2], as discussed in Section 2.4. In fact, a notation related to data flow diagrams is defined in [2], and it should be noted that data flow diagrams may be considered a constrained version of Activity Nets.

Unless otherwise stated, the here defined semantics is the basic semantics of Activity Nets. The specializations required for Stochastic Activity Nets and Attributed Activity Nets are explicitly mentioned.

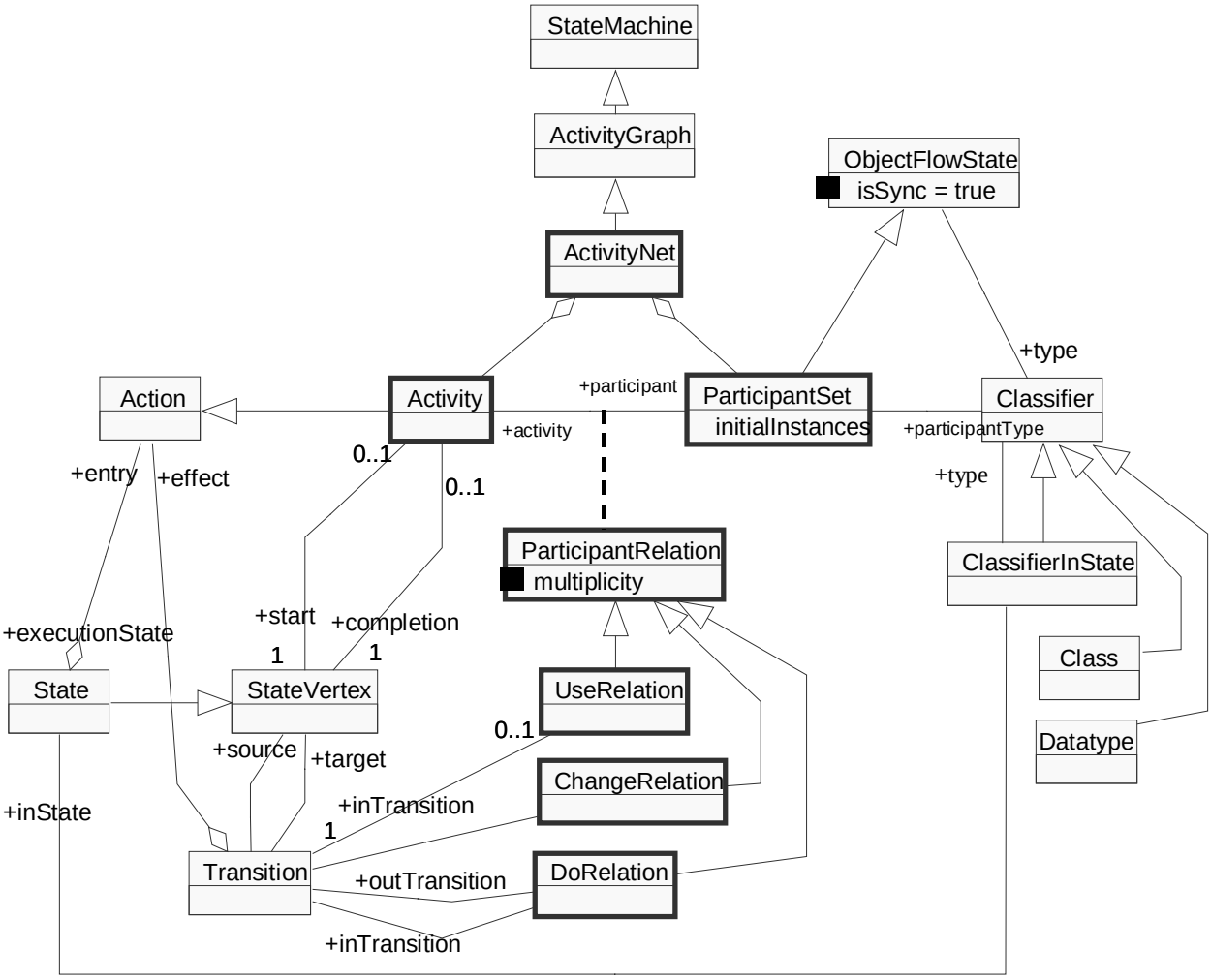

Figure 3: Abstract syntax of Activity Net concepts and their mapping to the concepts of UML Activity Graphs and State Machines

2.2. Stand-alone definition of Activity Nets

2.2.1. Basic semantic structure of Activity Nets

The static syntax of the concepts of Activity Nets are shown in Figure 1 and explained in the following.

Activity

From the semantic point of view, an Activity is a specialization of an Action as defined in UML (in other words, the base class of the Activity stereotype is the class Action). In general, an activity takes as input some participants from related ParticipantSets (UseRelation), produces as output some participants from other related ParticipantSets (ChangeRelation), and requires during its execution some participants from other related ParticipantSets (DoRelation). As such, an activity represents an action which changes the state of the system represented by the Activity Net, which is defined in terms of the instances of participant instances available at the different ParticipantSets of the Activity Net.

Associations

participant: through the designated ParticipantRelation, indicates the type of participants that are involved in the Activity and their relation with the activity. The ParticipantRelation must be of one of the following subtypes: UseRelation, ChangeRelation, or DoRelation.

(containment relation): Each activity is part of an ActivityNet.

ParticipantSet

A ParticipantSet represents a set (bag) of object instances which all belong to a specified class, which is defined by the UML Classifier pointed to by the *participantType*. This Classifier may be a ClassifierInState (as defined in UML Activity Diagrams), which indicates that the object instances are within a specified state (which must be defined in their associated StateMachine).

In the case of the basic semantics of Activity Nets, there is no need for the identity of the different object instances represented by a ParticipantSet; it is sufficient to count their number (since without attributes, they are all identical).

Associations

activity: through the ParticipantRelation, indicates the activities in which participants from this ParticipantSet are involved. The ParticipantRelation must be of one of the following subtypes: UseRelation, ChangeRelation, or DoRelation.

(containment relation): Each activity is part of an ActivityNet.

participantType: designates the Classifier which describes the type of participants contained in the ParticipantSet.

Attributes

initialInstances: the number of object instances initially contained within the ParticipantSet. In the case of Attributed Activity Nets, the attribute values of these initial instances must also be specified.

ParticipantRelation

A ParticipantRelation represents a relationship between an Activity and a ParticipantSet. This is an abstract concept, only the following three specializations occur in Activity Nets:

UseRelation: indicates that one or several participants of the ParticipantSet are used (consumed or transformed) by the Activity.

ChangeRelation: indicates that one of several participants of the ParticipantSet are created, or obtained in a new state through a transformation, by the Activity.

DoRelation: indicates that one or several participants of the ParticipantSet are required for performing the Activity; when the activity completes, these participants will return (unchanged) to the ParticipantSet.

Attributes

multiplicity: the number of participants from the ParticipantSet involved in the Activity (integer greater or equal to one).

2.2.2. Extensions for Stochastic and Attributed Activity Nets

The following additional attributes are defined in the case of **Stochastic Activity Nets**:

Attributes of *Activity*:

duration: this value indicates the duration of the execution of the Activity, once it is started. This attribute is in general a probability distribution (e.g. *uniform* between minimum and maximum values, a *fixed* value, *normal* distribution with given average and standard deviation, *negative exponential* or others). Therefore the duration of each particular instance of execution of the Activity may vary according to the distribution.

Note: In the case of an activity which has no relations, except a single ChangeRelation with a ParticipantSet, this duration attribute defines the rate by which objects in that ParticpantSet are created by the activity. In this case, a Poisson distribution with a given rate may be most appropriate to describe a random arrival process.

cost: this value indicates the cost of executing an instance of this Activity. A performance evaluation tool may determine the overall cost of various execution scenarios.

priority: this value indicates the relative priority of this Activity in respect to other Activities. The priority of Activities will be taken into account if, in a given system state, several Activities are enabled and in conflict, that is, only one of them can be executed (see Action Firing Rules in Section 2.2.4). However, more complex behavior patterns may be defined using specific values for the *interruptBehavior* (see below).

interruptBehavior: This attribute may take the following values:

- "no preemption": In this case the priority attribute is interpreted as described above.
- "normal preemption": In addition to the priority treatment described above, if an activity is enabled, except that a ParticipantSet which is related to the activity by a DoRelation does not have enough object instances, while another activity with lower priority, also having a DoRelation with the same ParticipantSet, is in the process of executing, then the execution of the latter activity will be preempted in order to let the former activity start its execution. The latter activity will be resumed as soon as sufficient objects become available in the given ParticipantSet.
- "continue after preemption with prolongation": The behavior is the same as for the value "normal preemption", except that the execution time of the preempted activity will increase by a given amount after preemption.
- "restart activity after preemption": The behavior is the same as for the value "normal preemption", except that when the execution of the preempted activity is resumed, the execution starts over from the beginning, that is, its duration after preemption is the same as it would have been without preemption.

Attributes of *ChangeRelation*:

percentage:

(a) In the case that the ChangeRelation has an exclusive-OR relationship with other ChangeRelations (see Section 3.2.3), then this percentage indicates the probability that this ChangeRelation will be chosen by the activity. Note: The sum of the percentage for all exclusive ChangeRelations should be equal to 100%.

(b) In the case that the ChangeRelation is not part of an exclusive-OR relationship, a value of the percentage smaller than 100% indicates that an object in the related PartcipantSet will not be created for all instants of execution of the activity. The value of the percentage indicates the probability that an object will be created.

cost: If non-zero, this is an additional cost which is added to the cost of the activity in the case that an object in the related ParticipantSet will be created. (Note: This applies to cases (a) and (b) above.)

In the case of **Attributed Activity Nets**, an EnablingPredicate may be associated with an Activity. Its meaning is explained in the Action Firing Rules of Section 2.2.4.

2.2.3. Well-formedness rules

There are no well-formedness rules for Activity Nets

2.2.4. Dynamic semantics

Activity

An Activity is a specialization of an Action. An Action is a specification of a transformation. In general, when executed, an action takes some input objects instances (or values), performs some processing and produces some set of output objects instances (or values). (A similar definition of Action can be found in Section 4.1 of [2]).

An Activity represents the possibility of executing its associated action once or several times, possibly in concurrency, according to the action firing rules described below.

ParticipantSet

A ParticipantSet represents a set of object instances which belong to the class specified by the Classifier designated by the *participantType* relation.

The number of objects in this set may change during the execution of the ActivityNet (as explained below). The initial number of object instances is indicated by the *initialInstances* attribute.

In the case of the basic semantics of activity nets and for Stochastic Activity Nets, the defined semantics is independent of the attributes of the object instances and internal state variables of the objects involved. Therefore one only needs to keep a count of the number of objects in each ParticipantSet.

In the case of an Attributed ActivityNet, the values of the attributes of objects may influence the possibility of action execution (through the associated enabling predicate). Therefore one has to keep track of the different object instances that reside at each instance of time within the different ParticipantSets. There may be different **options** concerning the order in which the object instances will be considered as input for actions to be executed. The default option is FIFO ordering. Other options may be nondeterminism (no order specified), or priority-based ordering.

UseRelation

The presence of a UseRelation means that one or several objects from the ParticipantSet designated by the *participant* are used as input by the action of the Activity designated by the *activity*. The number of objects taken as input is indicated by the *multiplicity* attribute. When the execution of this action is started these objects are taken as input and are removed from the ParticipantSet.

ChangeRelation

The presence of a ChangeRelation means that one or several objects belonging to the class indicated by the type of the ParticipantSet designated by the *participant* are produced during the execution of the action of the Activity designated by the *activity*. These objects are outputs from the action and will be placed within the ParticipantSet when the execution of the action completes. The number of these objects is indicated by the *multiplicity* attribute.

DoRelation

The presence of a DoRelation means that one or several objects from the ParticipantSet designated by the *participant* are involved in the action of the Activity designated by the *activity*. The number of these objects is indicated by the *multiplicity* attribute. This means that these objects are removed from the ParticipantSet when the action starts its execution and are placed back into the ParticipantSet when the action completes.

Action Firing Rules

An execution of an Action of an Activity may start when all ParticipantSets which are related to this Activity by a UseRelation or a DoRelation contain a sufficient number of object instances, that is, the number of object they contain must be equal or larger than the multiplicity of the corresponding ParticipantRelation. In this case it is said that the Activity is **enabled**.

Note: In a given state of an ActivityNet at a particular instant during its execution, there may be several actions that are in their execution state. Several of these may belong to the same Activity (representing parallel executions of different instances of this activity). At each instant in time, there may be zero, one or several Activities that are enabled.

In the case that several Activities are enabled, it is possible (in certain cases) to execute the actions of several activities in parallel. In other cases, there is a **conflict** between two or more Activities because they share the same input ParticipantSet (Use or Do relationship), and there are only sufficient object instances for starting one of the activities. In the latter case, a nondeterministic choice is made between the execution of these two Activities (unless priority considerations come into play, see Section 2.2.2).

In the case of Stochastic Activity Nets, one assumes that, as soon as one or several Activities are enabled, at least one of their actions starts its execution (no waiting of an activity in an enabled state). The considerations concerning priorities and preemption are explained in Section 2.2.2.

In the case of Attributed Activity Nets, there is an additional condition for starting the execution of an action: The enabling predicate of the action must be satisfied by the objects that participate in the actions as input through the UseRelations or through the DoRelations.

2.3. Mapping Activity Nets into UML Activity Graphs

2.3.1. A note on the use of the UML extension mechanisms

Following the defined extension mechanisms of UML for the definition of this profile, we should define the concepts of Activity Nets as UML stereotypes. Given the semantics of Activity Nets described above and the proposed mapping described below, it appears to be natural to define an Activity as a stereotype on the UML base class Action, a ParticipantSet as a stereotype with the base class ObjectFlowState and a

ParticipantRelation as a stereotype with the base class Relationship.

The attributes of the concepts Activity, ParticipantSet and ParticipantRelation described above should be formally defined as UML TaggedValues, since UML stereotypes are not allowed to define new attributes. We use nevertheless the notion of attributes for this purpose because this seems to be a more natural description of the concept.

The notation for the ParticipantRelations UseRelation and ChangeRelation is identical to the notation for the Dependency specialization of its Relationship base class. In the case of a DoRelation, it is very similar. The notation for the Activity and ParticipantSet is the same as for its base class. In the examples given in this paper (Figure 2, etc.), we have not shown the string "<stereotype>" as foreseen according to the UML conventions. The reason for this omission is that we think that this string is not required in order to avoid ambiguities, because the context should make clear that the given model diagram is an Activity Net and not an Activity Diagram.

2.3.2. Static structure of the mapping

The conceptual mapping of Activity Nets into UML Activity Graphs is shown in Figure 3 and further explained in the following. As an example, Figure 4 shows the Activity Diagram obtained from the example of Figure 2 when the here described mapping is applied.

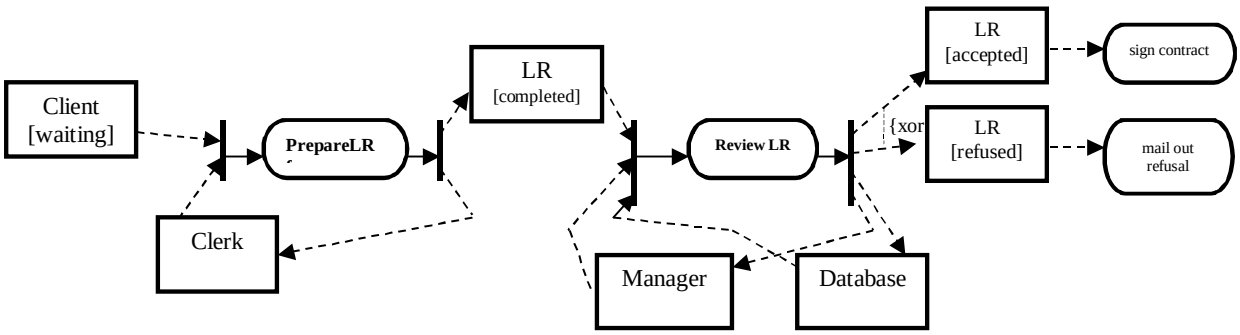

Figure 4: UML Activity Diagram equivalent to the Activity Net shown in Figure 2

ActivityNet

An ActivityNet is a specialization of an Activity Graph, which in turn is a StateMachine.

Activity

An Activity is a specialization of an Action as defined in UML (in other words, the base class of the Activity stereotype is the class Action). More specifically, an Activity is mapped to the entry action of a State which is designated by the *executionState* of the Activity (a relation of its parent concept Action). In addition, an Activity also has two Pseudostates designated by *start* and *completion*. They are *join* and *fork* states, respectively, and control the dynamic execution semantics of the activity. They represent the state before the execution of the activity and the state when the activity is completed,

respectively. The *executionState* of the Activity, as the name indicates, represents the state when the action is in the process of execution. There are also two transitions, namely from the *start* state to the *executionState* and from the *executionState* to the *completion* state. These transitions are not shown in Figure 3.

In a sense, each Activity is therefore mapped into three states in the corresponding Activity Graph with two transitions between them, as shown in the example of Figure 4.

From the notational point of view, an Activity is represented by the symbol representing an ActionState symbol as defined for UML Activity Diagrams. This ActionState may be considered to represent the *executionState* of the Activity.

Associations

start: designates a Pseudostate which represents a waiting state before the activity is executed. This must be a *join* state.

completion: designates a Pseudostate which represents the state when the activity is completed. This must be a *fork* state.

executionState (of Action): designates the state which represents the execution of the activity. The activity is the entry action of that state (as defined in [1]).

ParticipantSet

A ParicipantSet is a specialization of an ObjectFlowState as defined in UML Activity Graphs, which in turn is a specialization of State (in other words, the base class of the ParticipantSet stereotype is the class ObjectFlowState). The *participantType* of the ParticipantSet is realized by the *type* relation of the ObjectFlowState parent concept.

The *isSync* attribute (defined for the ObjectFlowState class) has the value *true*, and the *parameter* attribute is not used.

ParticipantRelation

A ParticipantRelation relates an Activity with a ParticipantSet. It is mapped to transitions in the Activity Graph, depending of the nature of the relation. A UseRelation is mapped to a transition designated *inTransition*, a ChangeRelation is mapped to a transition designated *outTransition*, and a DoRelation is mapped to two transitions, an *inTransition* and an *outTransition* (see below).

Associations

inTransition: designates a Transition from the ParticipantSet to the *start* Pseudostate of the Activity.

outTransition: designates a Transition from the *completion* Pseudostate of the Activity to the ParticipantSet.

2.3.3. Well-formedness rules

(A) Rules already mentioned in Section 2.3.2

(1) The *start* StateVertex of an Activity is a *join* Pseudostate.

(2) The *start* StateVertex of an Activity is connected with by a transition with the State for which the Activity is the *entry* action.

(3) The state for which the Activity is the *entry* action is an ActionState according to the UML definition of Activity Graphs, however, the attributes of the ActionState are not used for Activity Nets.

(4) The *completion* StateVertex of an Activity is a *fork* Pseudostate.

(5) Each Activity has a separate *start* and *completion* state.

(6) The State for which the Activity is the *entry* action is connected by a Transition with the *completion* StateVertex of the Activity.

(7) The *isSync* attribute of a ParticipantSet is *true*.

(8) The *inTransition* of a ParticipantRelation has as *target* the ParticipantSet to which the ParticipantRelation relates.

(9) The *inTransition* of a ParticipantRelation has as *source* the *completion* Pseudostate of the Activity to which the ParticipantRelation relates.

(10) The *outTransition* of a ParticipantRelation has as *target* the ParticipantSet to which the ParticipantRelation relates.

(11) The *outTransition* of a ParticipantRelation has as *source* the *completion* Pseudostate of the Activity to which the ParticipantRelation relates.

(B) Other rules

(1) The State designated by the *inState* relation of a ClassifierInState which is the *participantType* of a ParticipantSet must be one of the states contained in the StateMachine which describes the dynamic behavior of the Classifier which describes the *type* of the participants in the ParticipantSet.

(2) If a given Activity has a UseRelation and a ChangeRelation with two ParticipantSets which have as *participantType* two ClassifierInState classes which belong to the same Classifier (but different states), then this Activity must be the *effect* of a Transition that exists in the StateMachine of the Classifier and has as *source* StateVertex the *inState* state of the ClassifierInState associated with the UseRelation and as *target* StateVertex the *inState* state of the ClassifierInState associated with the ChangeRelation.

2.3.4. Mapping of the dynamic semantics

The following paragraphs provide some explanation of the semantic aspects of the mapping described above and indicate some minor discrepancies between the semantics of Activity Nets as defined in Section 2.2.4 and the semantics obtained from this mapping.

As explained above, an Activity is mapped onto three states in the Activity Graph: a *start* state, an *executionState* and a *completion* state. The firing rules of the Activity are realized by the *start* StateVertex of the Activity, which is a *join* Pseudostate. The *inTransitions* of the Use and Do relations of the Activity lead into this *join* state. Therefore objects must be present in the ObjectFlowStates from which these *inTransitions* start before the transition leading from the *start* state to the *executionState* can be performed (according to the standard UML semantics of Activity Graphs).

The creation of the output object instances of the action correspond to the *outTransitions* which go from the *completion* Pseudostate of the Activity to the ParticipantSets related by a Change or Do relation.

It is noted that the definition of the semantics of Transitions for Activity Graphs [1] states that *fork* and *join* Pseudostates must be well-nested according to rule #2 for PseudoStates in Activity Graphs. This does not apply to Activity Nets. Also the semantics for guards given in that paragraph does not apply to Activity Nets.

It is also noted that according to the definition of Activity Graphs, the Action represented by an Activity should be the entry action of an ActionState, and not a State (as shown in Figure 3). We have chosen not to use an ActionState at this point, but instead the more general notion of a State, because the concurrency semantics defined for ActionStates is not appropriate for Activities.

A ParticipantSet is a ObjectFlowState and inherits the semantics from that construct. In ActivityNets, ObjectFlowStates are used as *synch* states. The semantics of ObjectFlowStates in [1] specifies FIFO ordering for the object instances located at a given ObjectFlowState. This is also the default semantics for Activity Nets, however, other semantic options are also foreseen for Activity Nets (see Section 2.4). In [1] it is mentioned that “An object flow state may specify the parameter of an operation that provides its object as output, and the parameter of an operation that takes its object as input”. This facility is not used for ActivityNet modeling.

Note: The dynamic semantics of Activity Nets is equivalent to the semantics of Petri nets if one considers that an ActivityNet is a Petri net. A Petri net consists of places and transitions which are connected by arrows. We assume the following mapping from the objects of ActivityNet to the places and transitions of a corresponding Petri net: (a) Each ParticipantSet of the net is a place of the Petri net, (b) each Activity with its *start* and *completion* Pseudostates is a transition of the Petri net, and (c) the *inTransition* and *outTransition* transitions of the ParticipantRelations in the ActivityNet correspond to the arrows leading into and out of (respectively) the Petri net transitions which correspond to the Activities related by the ParticipantRelations. As an example, Figure 5 shows a Petri net equivalent to the Activity Net of Figure 2.

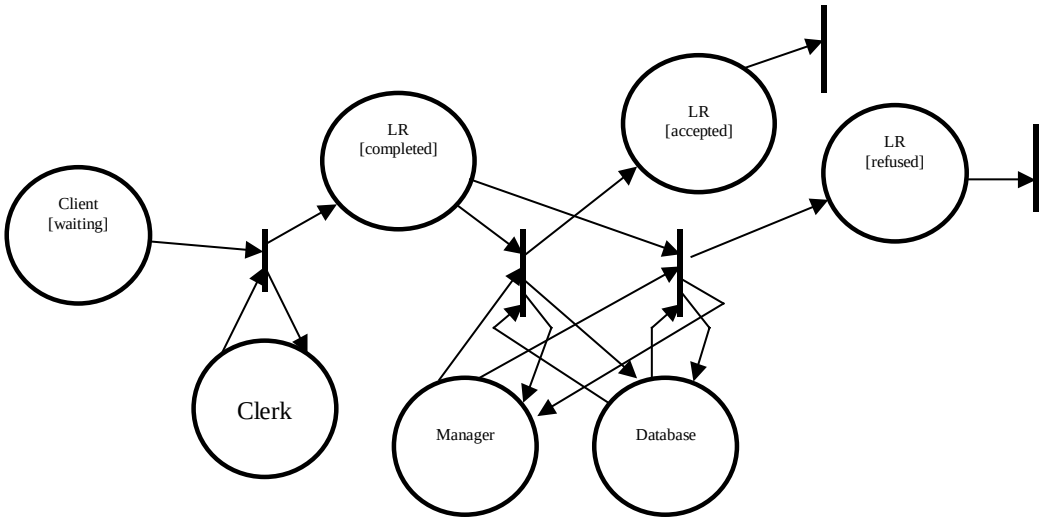

Figure 5: Petri net equivalent to the Activity Net shown in Figure 2 (Note: The five Petri net transitions (vertical bars) represent the activities "prepare LR form", "accept LR", "refuse LR", "sign contract" and "send out refusal", respectively)

2.6. Towards an alternative mapping of these concepts using the action semantics defined in [2]

The section 4.1 of [2], entitled Action Specification, describes the following concepts which are very relevant for Activity Nets: an action, inputs and outputs to an action (called pins), data flow, control flow, and the notion of a composite action which is composed out of other actions which are related through data flow. The other described concepts, procedures and local variables, are not very relevant to Activity Nets.

The following parallel can be made with the concepts of Activity Nets:

(1) An Activity is a specialization of a UML Action.

(2) The input and output pins of an action correspond to the Use and Change relations of an activity. A DoRelation may be modeled by two relations, one Use and one Change relation.

(3) A data flow relation between two actions corresponds to a ParticipantSet which is related to the first action by a UseRelation and to the second by a ChangeRelation.

(4) A data flow relation between two actions is usually associated with a data type or object class which describes the type of information flowing between the actions. This corresponds to the Classifier which represents the ParticipantType of the corresponding ParticipantSet.

(5) Control flow between actions can be represented by the direct sequencing notation of Activity Nets.

(6) The explanation of action substructure is similar to the activity substructure discussed for Activity Nets and corresponds to the “port” notation mentioned there. In fact, the notion of “pins” is similar to the “ports” proposed in [3].

The main conceptual differences between the notion of actions described in [2] and Activity Nets are the following:

(a) Activity Nets are not concerned with the detailed execution semantics which is described in the other sections of [2].

(b) Although not explicitly mentioned, it seems that for data flow diagrams one is not much concerned with the possibility of concurrent executions and the possibility of storing the intermediate results which are represented by data flow relations (such as this is done by ParticipantSets which explicitely represent a set of intermediate objects).

(c) Data flow diagrams are primarily intended for describing information processing, where it is trivial to copy the output of one action for input to many other actions. This is not necessarily true for the results of activities which may represent objects which would be expensive to duplicate. The objects of ParticipantSets often represent resources, the nature of which does not allow easy duplication.

Nevertheless, it seems that there are many conceptual communalities between composite actions, data flow diagrams and activity nets. Therefore it could be interesting to unify the conceptual framework for defining these concepts and notations. This would allow to define data flow diagrams as a special case of activity nets.

3. Notations

The presentational notation for Activity Nets proposed here is very closely related to the notation of Activity Diagrams as defined in UML [1].

It is noted that an alternate representation, described in [5] (see also Annex A), has been used extensively by DMR and its customers and is supported by an automated simulation tool.

In the following, we first define the representation of the Activity Net concepts described in Section 2. Then, in Section 3.2, we define a number of useful concepts the semantics of which can be defined by providing a translation into the semantics of basic Activity Nets. They may be considered to be shorthand notations.

Finally, in Section 3.4, we comment on the combination of Activity Nets with other UML diagrams in order to provide a full characterization of the system to be modeled.

3.1. Notations for basic Activity Nets

A diagram representing an activity net includes graphical symbols for the Activities, ParticipantSets and ParticipantRelations contained within the ActivityNet.

An Activity is represented in the form of an Action State symbol (see [1], Section 3.85), which may be thought of as representing the State which has the Action of the Activity as *entry* action. Conceptually, this graphical symbol also represents the *start* and *completion* StateVertexes of the Activity. The name written in the symbol, conceptually, represents the name of the action of the Activity. It is also the name of that Activity.

A ParticipantSet is represented in the form of an object flow symbol (see [1], Section 3.89), which represents the ParticipantSet considered as an ObjectFlowState. In the case that the associated object *type* is a ClassifierInState, the symbol for an *object in state* is used. The name written in the ParticipantSet symbol is the name of the ParticipantSet. In many cases, it will be equal to the name of the Classifier (or ClassifierInState) which is the associated object type. However, in certain cases, this name may be more specific, such as for instance “machine part of type X located near the assembly machine” where the “assembly machine” is one of the participants in the “assemble machine” activity.

The UseRelations and ChangeRelations are represented in the form of object flow symbols (see [1], Section 3.89), that is, in the form of a dashed arrow from the ParticipantSet to the Activity or in the opposite direction, respectively. The DoRelations are represented in the form of object flows in both directions, that is, in the form of a dashed line with arrows on both sides.

An example of an Activity Net is shown in Figure 2.

The representation of an Activity Net is very similar to an Activity Graph. However, the following differences should be noted:

(a) An activity net has no starting and ending states. The dynamic process described by an activity net either starts by an activity of the net that does not have any input flow, or the initial number of object instances in the different ParticipantSets allow a certain amount of processing to be done. (Note: A notion similar to starting and ending states is introduced for ActivityNets in the form of an interface, as discussed in Section 3.2.5).

(b) The fork and join Pseudostates shown in activity graphs are not explicitly shown in activity nets. These states are logically included in the Action State symbols, which simplifies the appearence of the activity net. For comparison, Figure 4 shows the activity diagram corresponding to the activity net shown in Figure 2.

3.2. Useful shorthand notations

3.2.1. Direct sequencing of activities

Sometimes a ParticipantSet has a ChangeRelation with one Activity and a UseRelation with another. In this case, the object instances in the ParticipantSet are created by the former activity and consumed by the latter. The ParticipantSet therefore represents an sequencing for the order in which these activities can be executed.

If the nature of the ParticipantSet (and in particular, the type of its object instances) is of no interest for the modeler and only the sequencing between the activities is intended to be modeled, it would be convenient to omit the ParticipantSet from the activity net. For this purpose a shorthand notation for defining a **sequential order of execution** between two activities is introduced. The notation consists of a solid arrow between two Activities and is a shorthand notation for a ChangeRelation from the first Activity to a (virtual) ParticipantSet and a UseRelation from that ParticipantSet to the second Activity.

It is noted that this notation may also be interpreted through a mapping to activity graphs as discussed in Section 2.3. The arrow representing sequential order of execution may be considered to represent a direct transition between the *completion* StateVertex of the first Activity and the *start* StateVertex of the second Activity, which corresponds to the semantics of a solid arrow transition in activity diagrams.

An example of the use of this notation is shown in Figure 6.

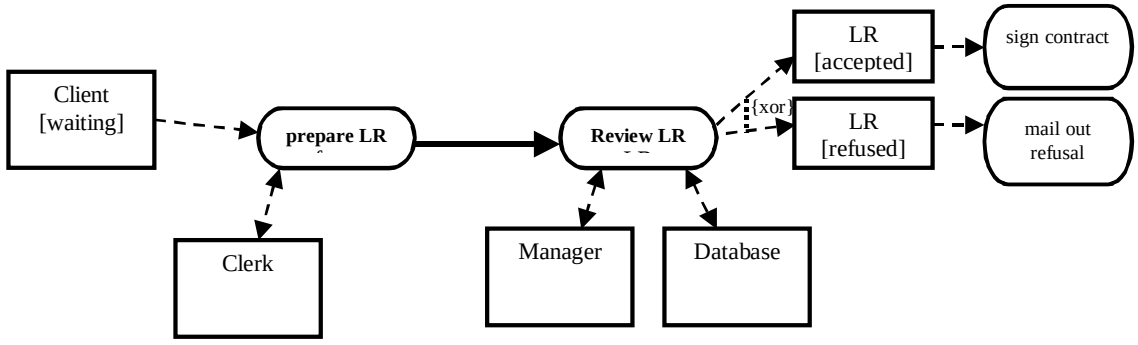

Figure 6: Activity Net showing directly the sequential ordering of activities (Note: This net is equivalent to the net shown in Figure 2, except that the ParticipantSet "LR [completed]" is not shown)

3.2.2. Swimlanes

The idea of swimlanes (see [1], Section 3.88) is to partition the diagram into a number of vertical bands (called swimlanes) which are associated with certain components (or organizational units) of the modeled system. The idea is that the component is involved in (or responsible for) all activities that are represented by symbols in the vertical band belonging to that component.

A generalization of swimlanes is proposed in Use Case Maps [9], where components are represented by rectangles which may be distributed over the diagram in an arbitrary manner (in contrast to swimlanes which are always represented as vertical bands). The actions defined in Use Case Maps may be shown within the contour of a component which means that the component is responsible for the execution of the action.

The same semantics can be modeled with activity nets by introducing a ParticipantSet for each component and introducing a DoRelation between each component (represented by the ParticipantSet) and the activities in which the component is involved.

Therefore, the swimlane notation defined in [1] may be used for activity nets and should be interpreted as a shorthand notation as follows: Each swimlane is an abbreviation for a new ParticipantSet with DoRelations to each Activity represented within that swimlane.

As an example, Figure 7 shows an activity net with swimlanes which is equivalent to the activity net shown in Figure 2.

In a similar manner, the component notation of Use Case Maps could be combined with activity nets.

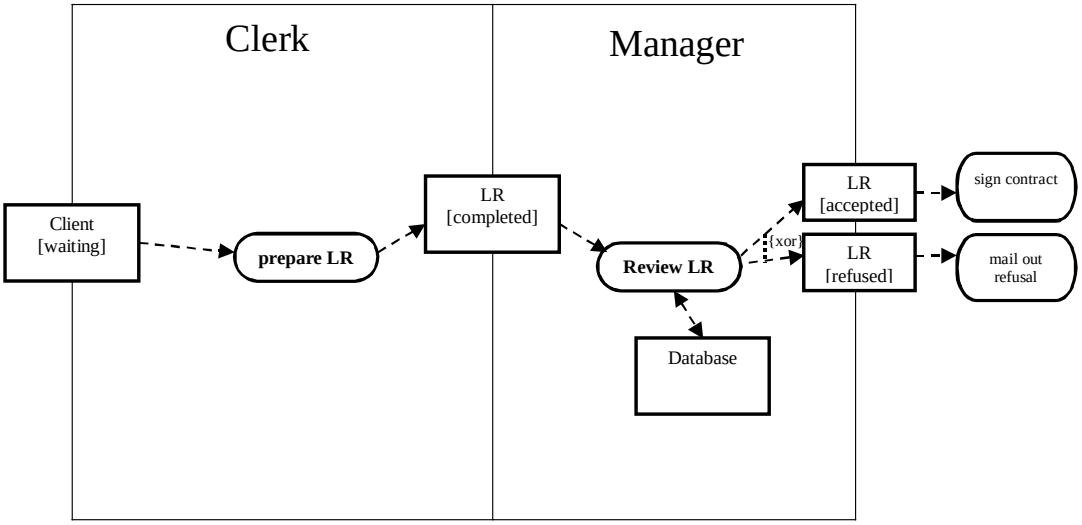

Figure 7: Activity Net with swimlanes equivalent to the net shown in Figure 2

3.2.3. Alternative execution results

It is often convenient to consider that a given Activity may have different results depending on parameters of the inputs or depending on unpredictable situations. Often one distinguishes between the normal result of an activity and the exceptional (or abnormal) case. An example is shown in Figure 2, where the resulting loaN request may be accepted or refused.

If a given Activity may provide mutually exclusive alternate outputs to different ParticipantSets, then this fact should be indicated on the ChangeRelations by the notation shown in Figure 2 which is taken from UML (see example in [1], Section 3.41.6). Note that conditions for the choice between the alternatives may be indicated in the form of enabling conditions. The choice of the output will in general lead to different follow-up activities. This corresponds to alternative execution paths.

Such a “choosing” Activity can be modeled by two “normal” Activities (as defined in Section 2.2), one producing only the first output and the other only producing the second

one. Both of these “normal” Activities would have the same UseRelations and DoRelations with other ParticipantSets as the “choosing” Activity. For given input object instances, only one of these Activities will be executed and will provide its specific output. The choice may depend on some enabling conditions which may be functions of the parameters of the input objects.

Therefore the notation for “choosing” activities can be considered as a shorthand notation, representing in fact two normal activities leading to two exclusive results. As an example, Figure 8 shows the semantics of the ActivityNet of Figure 2 in terms of the basic semantics described in Section 2.2 (without using the construct of alternate execution path).

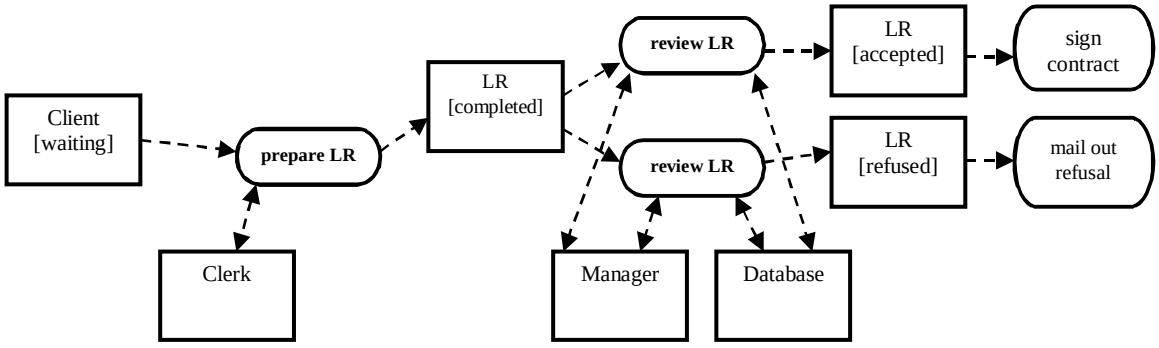

Figure 8: An Activity Net equivalent to the one shown in Figure 2, not using the exclusive OR explicitly

3.2.4. Distinguishing normal and alternative (exceptional) execution sequences

Activities with alternative results, as discussed in Section 3.2.3, lead to alternative sequences of activity executions. If we combine alternative results with the direct sequencing notation for activities described in Section 3.2.1, we get diagrams which show directly alternative execution sequences.

An abbreviated notation for normal and exceptional cases of alternative execution sequences is shown in Figure 9, which is equivalent to Figure 6, except that the use of thin and thick arrows indicates which sequence is the normal sequence (indicated by the think arrow), while the other sequences may be considered as exceptional cases. We note that in this case the exclusive Or is not explicitly indicated (but it is implied).

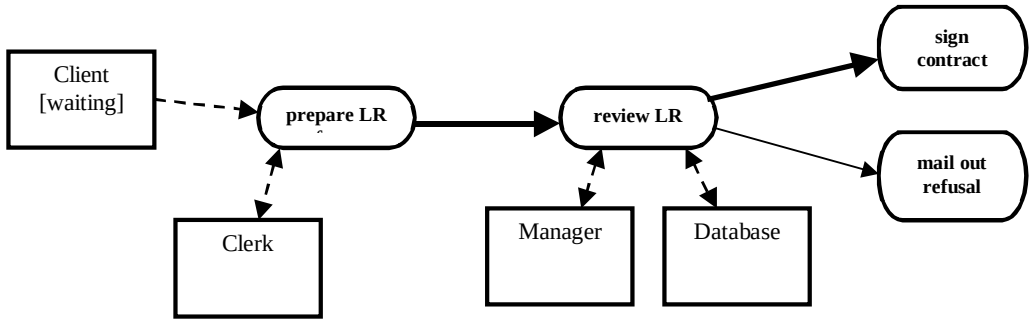

Figure 9: An Activity Net corresponding to the one shown in Figure 2, using direct sequencing with normal and exceptional alternatives.

3.2.5. Activity substructure

Sometimes it is desirable to describe the dynamic properties of a given system at different levels of detail. In the context of activity nets, this goal may be attained by considering that a given Activity within a given ActivityNet (called its context) can be described in more detail by describing its substructure. An example is shown in Figure 10. This notion is similar to a State of a state machine which may be defined as a submachine consisting of a collection of substates.

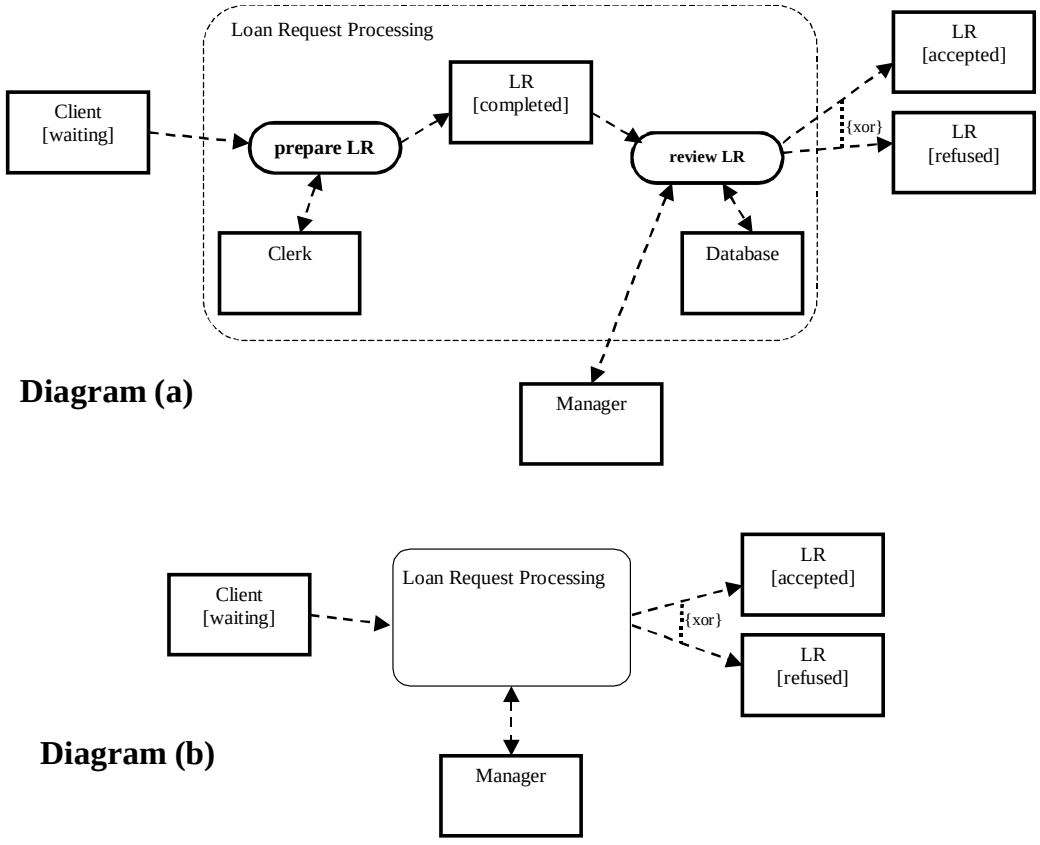

Figure 10: Activity substructure: The activity "Loan Request Processing" of the diagram (b) is shown in more detail in the diagram (a), both diagrams providing the same context.

***Definition**:* We call the ParticipantRelations of an Activity within an ActivityNet the **interface** of the Activity in the context of this ActivityNet.

***Definition**:* We call an **ActivityNet with an interface** a normal ActivityNet (as defined in Section 2.2) to which we add one or more ParticipantRelations for which the end-point designating a ParticipantSet is not assigned to any ParticipantSet of the ActivityNet. We call these partially assigned ParticipantRelations the *interface* of the ActivityNet.

In the case of the example of Figure 10, the interface of the activity "Loan Request Processing" consists of the following ParticipantRelations: a UseRelation with a "Client [waiting]", a DoRelation with a "Manager", and ChangeRelations with a "LR [accepted]" and a "LR [refused]", which are mutually exclusive .

Replacing an activity by an activity net: Given an Activity A within the context of an ActivityNet N and another ActivityNet N' with an interface, then A may be replaced in N by N' if there is a one-to-one mapping between the ParticipantRelations in the interface of A in the context of N and theParticipantRelations in the interface of N'. In this case we also say that N' is a refined (more detailed) view of A.

Inversely, we may consider a region of an ActivityNet N which is such that its border only cuts ParticipantRelations between some Activity which is inside the region and some ParticipantSet which is outside the region. In order to obtain a more simple (less detailed) model of the system, this region may be replaced by a single Activity and all ParticipantRelations that are cut by the border of the region will be connected to that new Activity. The resulting ActivityNet is called a less detailed view of N.

Concerning the notation for representing the interface of an ActivityNet, there are different possible approaches such a the following.

(a) To use the notation for submachines of UML Statechart diagrams: A notation similar to Submachine States of UML ([1], Section 3.82) lead to examples of the form shown in Figure 11. This notation has the disadvantage that it does not support a black-box view for the less detailed description; one has to refer to certain "components" of the activity, which is in conflict with the notion of abstraction.

(b) To define the interface through the introduction of "ports", "pins" or other placeholders: The ports introduced for RT-UML [3] allow for a black-box view at the abstract level and provide sufficient information for a proper checking of the "interfaces". A similar concept, called a "in", has been introduced in the draft of the UML Action Semantics [2]. We suggest to identify each ParticipantRelation of the interface by a "port", and obtain the example shown in Figure 12.

(c) To have no specific notation for the interface: In this case the diagram showing the detailed view of an Activity A in the context of an ActivityNet N must include all elements of N which make up the immediate context of A. This approach has been taken in [5].

The approach (b) is preferable, since it provides for a clear definition of the interface of an activity and therefore simplifies the checking of replacement compatibility and promotes a black-box view of activities.

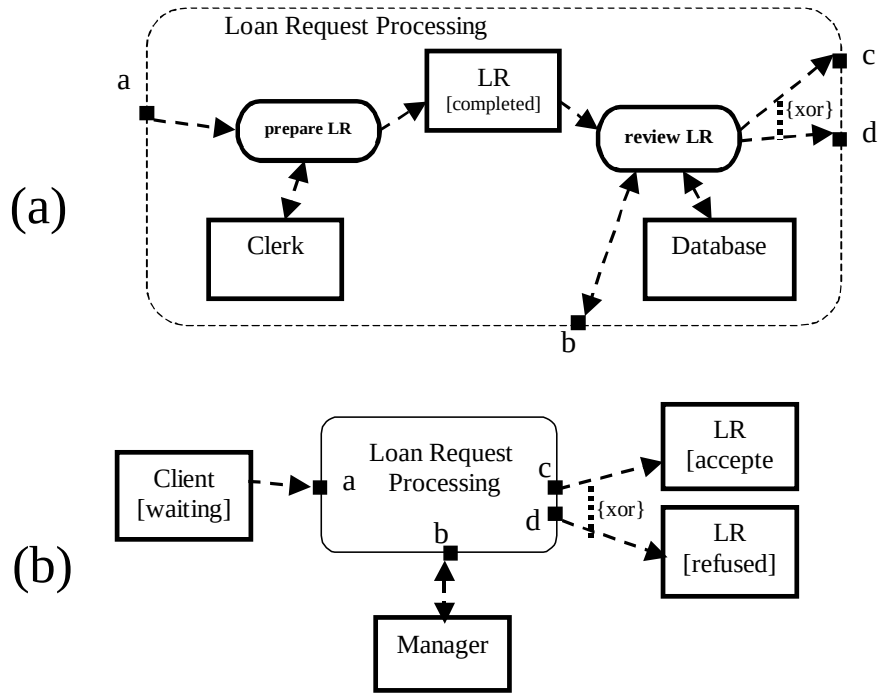

Figure 11: Activity substructure notation taken from UML Statemachines: Same example as in Figure 10. Diagram (a) shows the activity substructure and diagram (b) shows how the activity can be used in its context.

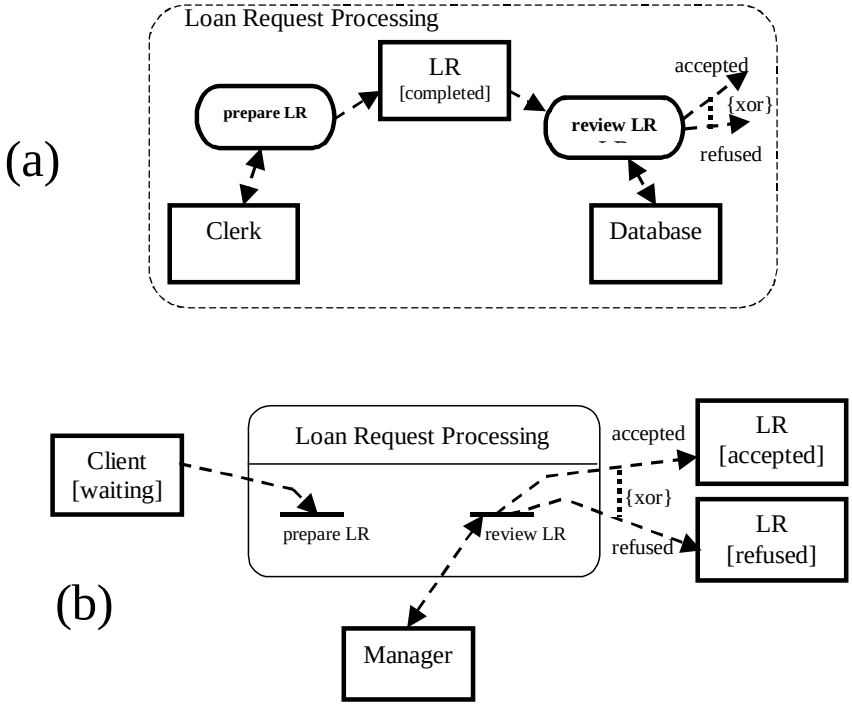

Figure 12: Activity substructure notation using ports: Same example as in Figure 10. Diagram (a) shows the activity substructure (including ports a, b, c and d) and diagram (b) shows how the activity can be used in its context.

3.3. Using activity nets with other UML diagrams

For describing a given workflow architecture, activity nets are usually complemented with UML Class Diagrams to describe the different types of Participants that are part of the workflow architecture. Depending on the application domain, different class hierarchies may be appropriate. For example, DMR's OPAL modeling guide [5], describes in Chapter 4 a hierarchy of Participant types which are suitable for the modeling of organizational architectures and workflows. A different set of object classes is defined in [13] for use in descriptions of software engineering processes. Another example is a rudimentary set of participant types defined in the UML Profile for Business Modeling (see [1], Chapter 4, Part 2).

Sometimes it may also be useful to use Class-Relationship diagrams to show containment relations, dependencies, transformation rules etc. (in addition to the Participant Relations already included in the Activity Nets).

State Diagrams describing the dynamic behavior of the participant object classes may also be included. However, they are already partly implied by the Activity Nets if the latter use ClassifierInState information for characterizing the objects in the ParticipantSets. Complete state diagrams seem to be more suitable for more implementation-oriented system designs which will be developed during a later phase of the system development process.

References

[1] OMG Unified Modeling Language Specification, Version 1.3, June 1999.

[2] UML Action Semantics (Draft), Action Semantics Consortium, 29 Jun 2000.

[3] UML-RT: A Profile for Modeling Complex Real-Time Architectures, Bran Selic, ObjecTime, Draft Version Dec. 12 1999.

[4] White Paper on the Profile mechanism, Analysis and Design Platform Task Force, OMG Document ad/99-04-07.

[5] OPAL Modeling and Simulation, DMR Architecture Lab, Technics Guide, Part 1.

[6] Conceptual Framework for Analysing Business Infrastructure as an Integrated Dynamic Model, by F. H. Stanley, Research and Development, DMR Group Inc., Montreal, 1992?

[7] A new approach to architectureal modeling and dynamic analysis of information systems and business processes, by G.v. Bochmann, A.C. Debaque, R. Dssouli, A. Jaoua, R.K. Keller, N. Rico and F. Saba, Technical Report, CRIM, Montreal, Dec/ 1992.

[8] The Macrotec Toolset for CASE-based Business Modeling, R.K. Keller et al., CRIM, Montreal, published ? 1994?

[9] On the Extension of UML with Use Case Maps Concepts, by D. Amyot (University of Ottawa) and G. Mussbacher (Mitel, Ottawa), to be presented at the UML conference, Oct. 2000.

[10] K. Jensen, Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use - Volume 1, EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1992.

[11] K.S. Trivedi, J.K. Muppula, S.P. Woolet and B.R. Haverkort, Composite performance and dependability analysis, Performance Evaluation, 14 (3-1), pp. 197-215, 1992.

[12] OMG document: Workflow Management Facility, Convenience Document combining dtc/99-07-05 and dtc/2000-02-03 (WF RTF 1.3 Report), 14 February 2000.

[13] OMG document ad/2000-05-01: Software Process Engineering (SPE) Management RFP, Initial Submission, Fujitsu/DMR.