UML BUSINESS MODELING PROFILE

Audris Kalnins, Janis Barzdins, Edgars Celms*

1. INTRODUCTION

At the time of writing this paper, UML 2.0 has reached its near final status, no significant changes are expected. On the other hand, various aspects of business process modeling have regained their significance for information system design. Especially it is due to clear statement that business process model is the main part of the Computation Independent Model – the first step in Model Driven Architecture (MDA) application [1]. UML 2.0 is supposed to cover all MDA steps.

The goal of the paper is to investigate how adequate is the new UML 2.0 for business process modeling, especially from the semantics point of view. Our analysis will be based on the current proposal for UML Superstructure [2] (08.2003) by the OMG UML 2.0 finalization task group (FTF) – the document, on the basis of which the final UML 2.0 version is being prepared.

The analysis in the paper shows that the behavioral semantics of UML 2.0 Activity diagrams still does not fully comply with the traditional business process modeling semantics. At the same time, UML 2.0 contains the profile mechanism, which permits to adapt the semantics to specific requirements, in our case business modeling. The main result of the paper is a proposal for Business Modeling profile. This profile to a great deal adjusts UML semantics to business process modeling requirements – the need for a business transaction concept and semantically correct process resource definition.

The paper starts with a brief description of the traditional business modeling and its semantics. Then in section 3 the new activity diagram semantics is analyzed and its deficiencies for business modeling shown. Next sections show how these problems could be solved by profiles – the missing transactions and then the resource management.

It should be mentioned that currently there are very few papers evaluating UML 2.0 from the process modeling point of view – the only known one is [3], and it is more from a workflow point of view.

*University of Latvia, IMCS, Raina bulv. 29, Riga, LV-1459, Latvia
Audris.Kalnins@mii.lu.lv, Janis.Barzdins@mii.lu.lv, Edgars.Celms@mii.lu.lv

2. TRADITIONAL BUSINESS PROCESS MODELING AND ITS SEMANTICS

The area of *business process modeling* (BM) has grown out of activities such as Business Process Reengineering (BPR) – a buzzword of mid nineties and Business Process Management (BPM) – the current buzzword. One more current trend is business process definition for the specification of correct web service interaction – the goal of languages such as BPMN[4]. The business process modeling area has its own preferences for modeling languages and tools. Though no one absolutely dominant language as in object modeling exists here, there are leaders in this area too. One of the most popular BM languages, supported by an equally named tool is ARIS by IDS Prof. Scheer [5]. According to Gartner reports [6], this is the most used BM tool. But another widely used BM language is IDEF language group (the most important of them is the language IDEF3 [7]). IDEF languages are supported by numerous tools, possibly the most popular among them is System Architect by Popkin Software [8]. But there are also many other business modeling languages and notations such as Rummler-Brache process charts (known for a long time, but described more formally in [9]), which are the base for business modeling in Oracle Designer and Proforma tool, QPR flowchart notation et al. The GRADE-BM approach [10] proposed by IMCS LU and Infologistik (with the participation of authors of this paper) should also be mentioned. Though each of the languages listed here has its own terminology and specific details in process diagrams, the flowchart-like process diagram syntax is quite similar in all of them. And it has much in common with the UML activity diagram notation. Besides pure modeling, most of BM notations provide also process simulation, required both for the goals of BPR and BPM.

Traditional business modeling languages [5,7,10] have much in common also from the semantics point of view. Their degree of semantics definition formality varies, but since most of the modeling notations support also process simulation, there must be at least internally defined absolutely precise process semantics. So we try to present these common business modeling semantics features on an example – at first informally and then in a more formal manner. The example – Fig.1 is actually a UML 2.0 activity diagram from [2], describing a simplified mail order servicing process, but it would look quite similar in any of the special business modeling languages [5,7,10] from the graphical syntax point of view.

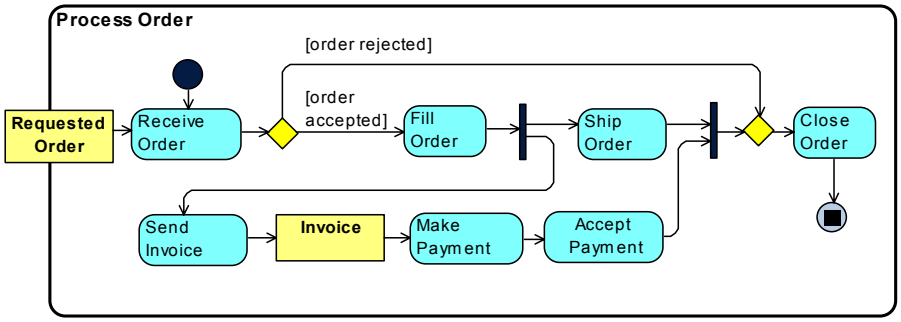

Figure 1. Business process example.

The process contains a number of elementary steps (activities, tasks,…), which have to be performed in a certain sequence to achieve the goal – to process an order. Each of these steps requires some (possibly varying) time to be executed. There are possible branchings in this sequence, and also, what is more important, some of the process steps

(`Ship Order`, `Make Payment`) can be executed concurrently. But there is more concurrency in this process. The Mail Order company actually services a lot of orders simultaneously, so there exist a lot of instances of this process simultaneously. Business modeling semantics typically assumes that the *same diagram instance describes all of these process instances*, each of which is in a different phase of readiness – some are executing a step, some are waiting for the next step to be done. Typically there are several concurrent instances of each activity (as far as the resources of the company permit – see more on resources in section 5). But at the same time from the logical point of view the processing of each order must occur independently – there should be no mixing of "data" for different orders.

The only place where this requirement can cause problems is where concurrent subprocesses of a process instance join. In the example, it is the vertical bar symbol expressing that only after an order has been shipped and the customer has made the payment for namely *that* order, the order can be closed. If the semantics would be considered only from the "common sense" point of view, no one would understand this in a different way. But for the formal semantics definition there may be problems. Due to the fact that the shipment time is dependent on distance, and customers pay for an order when they want (within the permitted interval), different orders overtake each other at this point – there may be many "shipment done" notices waiting for respective payments, which arrive in arbitrary order. For any formal semantics definition there should be a provision specifying that a shipment notice is matched only with the corresponding payment information (to avoid the need for an explicit data-based joining rule to be added by the modeler, which would be extremely awkward).

The most exact BM concept for this purpose is the *transaction* concept. In some form it appears in ARIS [5], but in the most explicit form it is provided in GRADE [10]. The idea is to define each "business transaction" instance of a process (servicing a mail order, in the example) being also an instance of the formal transaction associated to the process diagram. Each instance of such transaction then can have a formal unique identifier (TID), which is generated when the process instance starts. Each instance of the process element – a step, a message waiting to be forwarded etc. is labeled by the same TID. There are some natural rules how the TIDs are propagated though steps. But the most interesting part of the definition is for concurrent subprocess join points. There a default join rule is automatically added, saying that only subprocesses with equal TIDs may be joined. The exact formulation of TID rules depends on the specific syntax of the process notation, but in any case, they solve the problem described above. It should be noted that something similar exists in the world of workflow definitions, where various folder and subfolder IDs are used for similar purpose.

It should be noted that from the conceptual point of view the transaction bears some similarity to UML use case with respect to the activity which describes the behavior of that use case – it is a response to some external request or message (certainly, the syntactic context is completely different).

3. UML 2.0 ACTIVITY DIAGRAM SEMANTICS AND BUSINESS MODELING

The diagram type mainly used in UML for business modeling is the activity diagram. It is also the most modified diagram type in UML 2.0 (with respect to the current UML version) – with the goal of workflow definition and partially also business modeling in mind. But the second goal evidently has been to incorporate seamlessly the

Action semantics from UML 1.5, which has made the definition rather complicated. The second goal is not considered in this paper at all.

At first few comments on terminology and notations in UML 2.0 – to make the paper readable for those who have not thoroughly studied [2]. The complete activity diagram now corresponds to the `Activity` concept in the UML metamodel. The diagram area is encircled by a frame containing its name and additional attributes. The rounded rectangles it contains (the process steps) are now called `Actions`, in the example all of them actually are assumed to be `CallBehaviorActions` invoking equally named `Behaviors`, or more precisely, non-refined `Activities`. Thus in the BM area an elementary activity step – *action* now can be either a call of non-refined activity (described by its name and some informal description), or a call of another activity described by its own diagram. Control flows retain their syntax from UML 1.5. But object flows in addition to the old syntax (with an object symbol in the middle) have a new one with *object pins* at both ends (see the `Invoice` flow in the example). The pin where the flow starts is called the *output pin*, the opposite one is the *input pin*. The name of an object or pin specifies the type, instances of which can be placed in it. The new syntax is more adequate for the semantics definition. Both types of flow can have guard conditions on them. One more syntactic element is the *activity parameter* (input or output) symbol (the input parameter `Requested Order` in the example is presented as a box positioned on the activity frame). The types of available control nodes are the same as in UML 1.5, namely, decision, merge, fork, join; only the final node now has two subtypes: activity final (the old one) and flow final.

The same example from [2] is used (Fig. 2). The only difference from [2] is that the pin syntax is used for the object flow.

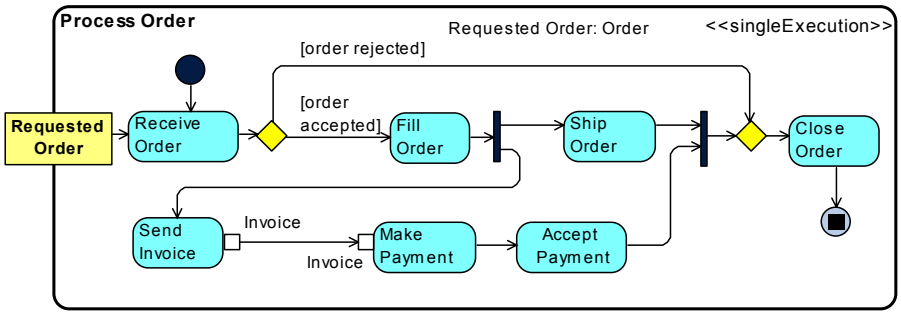

Figure 2. A complete example of the new activity diagram in UML 2.0

In order to achieve the traditional business modeling semantics, two execution mode specifiers now available in UML 2.0 must be used. Each activity definition can have a special attribute `isSingleExecution`, which for any activity representing a business process must be set to `true`. Then the corresponding keyword `<<singleExecution>>` appears inside the diagram frame. Only in this mode one instance of the diagram (one execution of the activity in terms of [2], see p.287, section (CompleteActivities)) is used to serve all process invocation instances – as it is normally required for real business process modeling, described in section 2. Each activity called by an action (elementary or refined) in turn can have an attribute `isReentrant` which must be set to `true` for business modeling (thus allowing many copies of this activity to

execute concurrently – an absolute must for business modeling). Since this attribute has no visible representation in diagrams, we assume it to be `true` for all examples.

As already stated, the formal semantics now is based on the theoretical model of Petri nets [11] and therefore inherently concurrent – the first bonus for BM applications. The Petri net model dynamics is based on token movements along the edges of a diagram. According to [2], a control token (without data) corresponds to any instance of control flow "flown" and an object token containing the respective object instance corresponds to an instance of an object flow. When an activity execution is to be started (the activity is invoked, in terms of [2]), a control token must be placed in the initial node (if there is such) and an object token of appropriate type in each input parameter.

To explain the situation with semantics definition for business modeling, we have to formulate the general execution (token movement) rules here. We present them in a more compact form with respect to [2], with only the aspects relevant to business modeling defined. To make the description easier to understand, we make one special assumption. Any "normal" (action-to-action directly) object flow having the UML 1.5 style with an object symbol in-between should mentally be converted to the new style with pins at both ends. "Multiconnected" object nodes (objects with the pattern of connected edges other than exactly one incoming edge and one outgoing edge) are considered to be buffers or data stores (both concepts appear also in business modeling). In addition, control flow edges outgoing from actions should also be thought as having an invisible *output* "*control*" *pin* at the start, for control tokens to reside. Similarly, a control flow edge entering an action is assumed to have an invisible input "control" pin, just to understand these token rules. A special case is the flow (object or control) edges entering or leaving the control nodes, there pins are not built. The typical situation in activity diagrams for business modeling is that output pins of actions serve as real queues for tokens – messages traveling in a business process.

Now the compacted token rules:

- Tokens can reside only in parameters, output pins (in special cases also input pins), buffers and data store objects (see later) and the initial node.
- A token from its place of residence can travel along a flow if the guard condition on it (if there is such) is true. But a token never moves if a downstream action does not consume it.
- An action execution is created when all its object flow and control flow prerequisites (i.e., for any entering edge) have been satisfied (an implicit join or default AND-triggering condition in terms of the business modeling). A flow prerequisite for an edge is satisfied when its input pin is offered a token – that is, the required token can travel from its place of residence along the edges to that input pin and no edge condition or control node along the path can prevent this. When an action is enabled, it accepts the required tokens (places them at the corresponding input pins) all at once, precluding them from being consumed by any other concurrent actions and thus avoiding deadlocks.
- If additional conditions for the action (if any) are true, the action execution really starts. It means that tokens are removed from the input pins and consumed internally. For business modeling, the action execution means the execution of the called activity – elementary one or refined by an activity diagram of its own. In the case of a refined activity, the consumed tokens are placed into input parameters (and/or initial node) of the refining diagram.

- An action continues executing until it has completed. When completed, an action execution offers object or control tokens on all its output pins and terminates. Any output pin now contains a new token in its queue, which can be used by subsequent actions. It should be noted that this is an implicit fork construct - several outgoing flow edges create a potential concurrency.
- If an edge is connected to a control node, no pin (real or invisible) is assumed at that end. Control nodes just define the rules how tokens can pass them – these rules will be defined separately for each node type. All these rules are in the form of possible token offers from input edges to output edges – actual token movements are determined by the acceptance by the final target – an action.
- Decision nodes are assumed to have outgoing edges with mutually exclusive guard conditions attached. The semantics definition is simple – any incoming token is forwarded to a single outgoing edge according to conditions.
- Merge nodes are used to merge together multiple alternate flows. Semantics is trivial – any one incoming token is forwarded to the outgoing edge.
- Fork nodes are used for explicit splitting (duplicating) of flows, they have one incoming edge and several outgoing edges. The semantics is simple – a copy of an arriving token is offered to any of outgoing edges.
- Join nodes express explicit synchronization of subflows in a process – an equivalent of AND-triggering. There can be several incoming flows of different types – both object and control and one outgoing. If there is a token offered on all incoming edges, the token is offered on the outgoing edge according to the following join rules:
  - If all the tokens offered on the incoming edges are control tokens, then one control token is offered on the outgoing edge
  - If some of the tokens offered on the incoming edges are control tokens and other are data tokens, then only the data tokens are offered on the outgoing edge.
- Tokens arriving at a flow final node are simply destroyed, but at an activity final node the complete activity execution is terminated (i.e., all other tokens in it are aborted) and output parameters (if any) are returned to the caller.

This concludes the precise semantics definition.

Thus the basic semantic requirements for business modeling via activity diagrams are now in place. But two problems in this simple example can be noted on the spot. It can be easily seen that there is no provision in activity diagram semantics for a transaction-like concept. This means the problem defined in section 2 can appear – the semantics definition does not preclude a control token generated by an instance of `Ship Order` to be joined with a control token from `Make Payment` for a completely different order at the join node. Another simpler problem is that typically the process definition in activity diagrams is ended via *Activity final* node (as in Fig.2). But according to the semantics just described the first *Order* reaching the final node would terminate all the other in-process ones. Here *Flow final* node could be used, but it is not adequate if the activity in Fig. 2 actually were invoked by an action – then it could not provide the return of output parameters (if there were such). The paper proposes solutions to both of these problems in the next sections.

We conclude the section with a short overview of another elements of UML 2.0 proposal, which are relevant for business modeling, and how they affect semantics.

There are two more action types – *AcceptEventAction*, with a subtype for time actions. Their use in business modeling could be as spontaneous start nodes – either for the whole process, or as enablers inside. They would be followed by a control flow, with simple semantics – the control token generated at appropriate moments.

The concept of central buffer permits object flows from several actions to supply objects to common storage, from which they may be taken by several flows (all flows of the same type). The data store node defines a similar behavior, but the flowing objects are assumed to be stored there permanently (as copies).

It should be noted that the described syntactic facilities cover the main execution control facilities to be found in typical business modeling notations.

4. NEW ACTIVITY SEMANTICS FOR BUSINESS MODELING (BUSINESS MODELING PROFILE)

In this section we propose a semantics amendment, which would solve the problems for join and final nodes, noted in the previous section in the context of using UML 2.0 activity diagrams for business modeling.

The proposed solution complies with UML standards for extension – it is a proposal for Business Modeling profile (see Fig. 3).

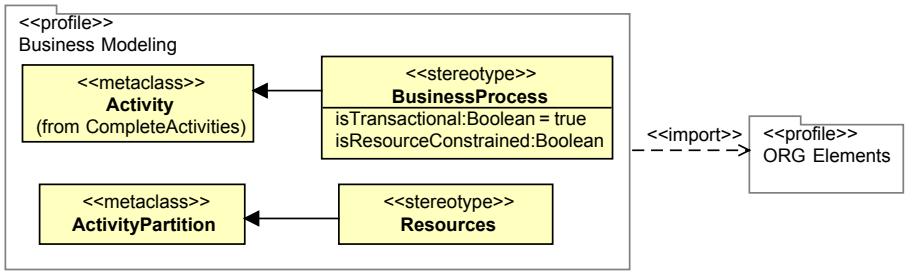

Figure 3. Business Modeling profile.

The goal of the profile is to define an execution semantics modification, based on the transaction concept in BM, described in section 2. To do it, we propose an activity stereotype - `<<BusinessProcess>>`. This stereotype adds a new Boolean attribute `isTransactional` with default value `true` to the metamodel class `Activity` (for `Complete Activities` package). The other elements in Fig. 3 will be explained in the next section. When an activity is transactional (i.e., the new attribute `isTransactional` has the value `true`), new token management rules described in this section are added to the standard rules described in Section 3. It only makes sense to make an activity transactional, when it has the `SingleExecution` property.

The concept of profile in UML 2.0 formalizes only the extensions to modeling syntax – what new element properties (metaattributes) are introduced by stereotypes. But for adapting the semantics semiformal *execution profiles* may be used, which "may tighten the rules to enforce various kinds of execution semantics" (see [2], p.287). The transactional mode of execution is namely the tightening of token flow rules.

The main idea for the new transactional mode is that any token (control or object) has to be extended by a new data element – *transaction identifier* (TID). It has a unique value for each activity invocation. The main use of TID is at join nodes and actions having several incoming flows (implicit joins). There a default join specification

(or default precondition) is provided which enables the required semantics – only those concurrent subprocess instances which have *originated from one process instance* can join.

The formal definition of the transaction mode execution is the following. A formal identifier (TID) is added to any token. The added rules for token management are the following:

- When an activity (i.e., activity diagram) is invoked, a new unique TID value is generated and assigned to the tokens placed in the initial node and all input parameters (i.e., to all tokens which are provided by the invocation in order to start the execution instance)
- When an action is activated by a single token (one incoming edge), the same TID is attached to any token placed by the action in its output pins after it has completed.
- When an action has several incoming edges the TIDs for all tokens to be consumed by the action must be the same. In other words, only a group of relevant tokens with equal TIDs is accepted as an execution enabler, other token combinations remain in their places of residence. The new condition can be considered as a default precondition. After completion the common TID is assigned to the output tokens.
- For a join node: a group of tokens arriving via its incoming edges can only pass the node if their TIDs are equal (it is a default join specification). If a new control token for the outgoing edge has to be generated, it also has the same TID value.
- For a fork node: all duplicates of the incoming token have the same original TID
- For decision and merge nodes: no special rules since tokens are simply forwarded
- For activity final node: in addition to incoming token destroyed and output parameter tokens (without TIDs) returned, all tokens having the same TID (as the incoming one) are destroyed and actions started by tokens having this TID are terminated. Thus every activity related to this transaction instance is terminated.
- For flow final node: no additional actions.

It does not matter whether the activity is actually a top one or a subactivity, the same TID rules are used. In other words, the refinement activity execution is completely "encapsulated" within the corresponding call action execution, without any possibility to make a non-structured return to somewhere else. Therefore the formal transaction rules for an activity execution need not to take into account the activity structuring.

The only special fact that should be added is that for edges having weight larger than one at joins (and actions with implicit joins) the TID equality is not required for tokens arriving on this edge.

It can be easily seen that the described rules provide the functionality expected in business modeling. Now, the shipment notices can only join with payment information, if they come from the same order. In addition, in the example in Fig. 2 it is completely correct now to end the activity with the activity final node.

5. ONE MORE SEMANTICS PROBLEM – PERFORMER SPECIFICATION

Traditional business modeling semantics actually has one more aspect not considered in section 2. It is the specification of required performers for each activity in process diagrams. In addition, there normally is something like an Organizational structure (ORG) diagram (see ARIS [5] or GRADE [10]) accompanying the process model, where the available performer resources are specified. This diagram typically has a tree structure (see Fig.4), with organizational units as upper level nodes and various positions in these units as leaves (sometimes more kinds of nodes are available).

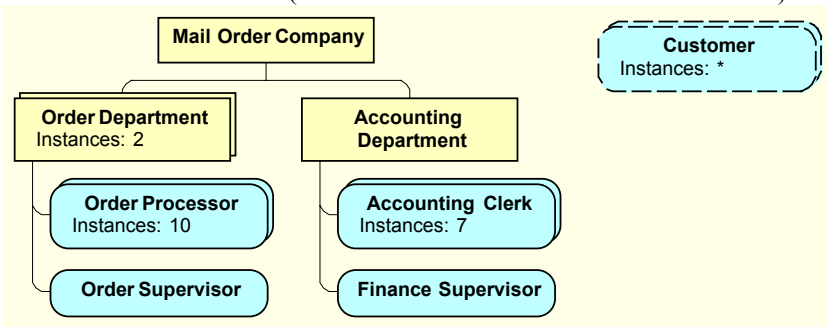

Figure 4. Organizational structure diagram example.

At each node the number of available instances can be specified. Typically the positions are specified as activity performers. With that information available, a more realistic resource-constrained execution of a process can be defined where an activity instance is started only when an instance of the required performer resource (or several resources) is free – i.e., not used by another activity instance. Namely, this is the model used for various BPR activities and simulation, only this way real bottlenecks in an organization can be found.

With the sophisticated *Activity Partition* concept UML 2.0 provides a lot in that direction. The example (slightly modified from [2]) in Fig.5 shows an acceptable for BM practice performer notation – the one frequently used in BM as an alternative to swimlane notation.

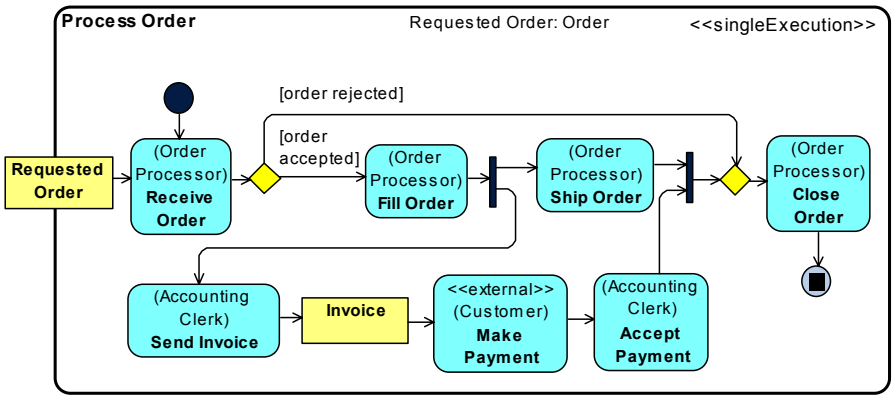

Figure 5. Performer specification in UML 2.0.

The Activity Partition concept permits various types of structuring – simple hierarchy, structuring along several dimensions etc. Activity partitions can represent (i.e., reference by name) various model elements – classes, instances, attribute values. It would be completely sufficient for BM purposes from the syntax point of view. The main problem is the lack of semantics – it is stated that partition specification can in no way influence the token flow in semantics definition.

It is comparatively easy to define one more extension to activity execution semantics which would coincide with the rules for traditional resource-constrained execution in BM semantics. The main problem is how to define the available resources. One of the solutions is to use a stereotyped class diagram as a substitute for Organizational structure diagram. Since in UML there is no simple way how to restrict the class instance count, a new property for instance count is added to these stereotypes. The proposed stereotypes form the `ORG Elements` profile (see Fig. 6), which is part of (imported into) the proposed earlier `Business Modeling` profile.

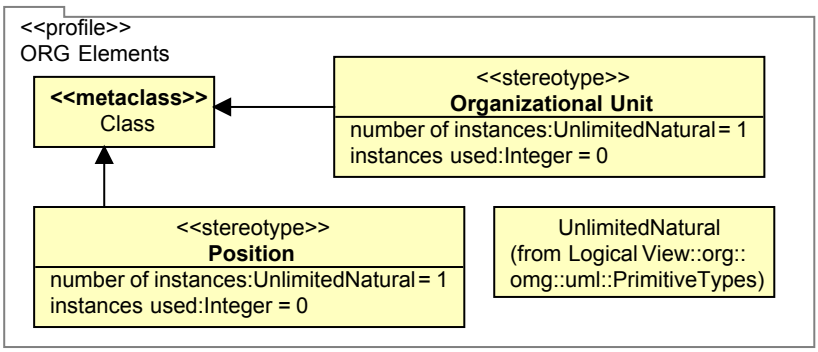

Figure 6. ORG Elements package.

There could be more types of elements in the profile, and there could be more properties for stereotypes to have. But the items defined here are sufficient to demonstrate the idea. Fig. 7 shows an example of ORG diagram equivalent to Fig.4 in the form of a class diagram.

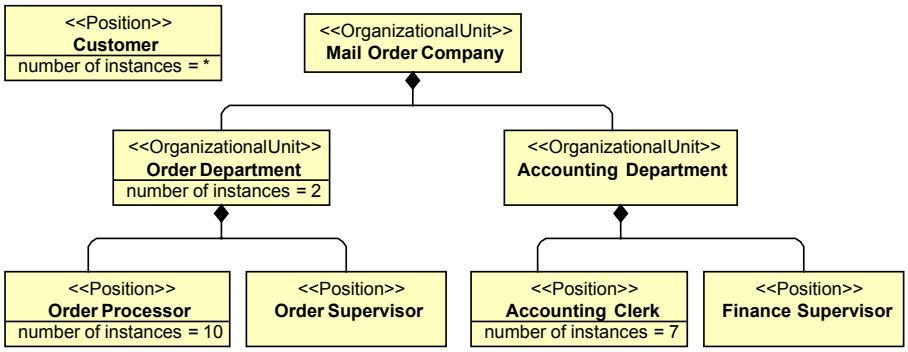

Figure 7. ORG elements as class diagram.

The `number of instances` property of the stereotypes here is shown directly in class symbols (the notation proposed by [2] is not acceptable in practice). The other property `instances used` is required only for semantics definition. As it is

typically assumed in BM semantics, numbers of instances along the hierarchy levels multiply, i.e., there are 20 instances of `Order processor` in the example. If the number of instances is omitted, the default value 1 is assumed.

Now the performer specification rules can be defined. The activity partition `Resources` with `isDimension = true` could be defined, subpartitions of which can only represent classes with the stereotype `<<Organizational Unit>>` or `<<Position>>`. Thus the elements within this dimension would be treated as true activity performers with the semantics typical to BM. The `Resources` partition stereotype also had to be the part of the Business Modeling profile (with the abovementioned constraints included).

The last thing to be defined is one more Boolean attribute `isResourceConstrained` for the `<<BusinessProcess>>` stereotype of Activity in the Business Modeling profile. If this new attribute is true, two more rules are added:

- If an action execution is to be started, then if the action is within a partition being a subpartition of `Resources`, then the corresponding class is found and it is checked whether the value of `instances used` is less than the `number of instances`. If yes, then the action execution is started and `instances used` is incremented by 1. If no, the action is not started (all the corresponding tokens remain in input pins).
- When an action execution completes, the `instances used` of the corresponding `ORG element` class is decremented by 1. In addition, any activity, whose partition references the same class, is checked whether its input pins contain a set of tokens. The first one such found is activated (according to the rule one here).

This completes the semantics definition for the resource-constrained case. Fig. 8 shows the final version of the example, with the both new execution modes set and with the corresponding ORG elements specified in Fig. 7.

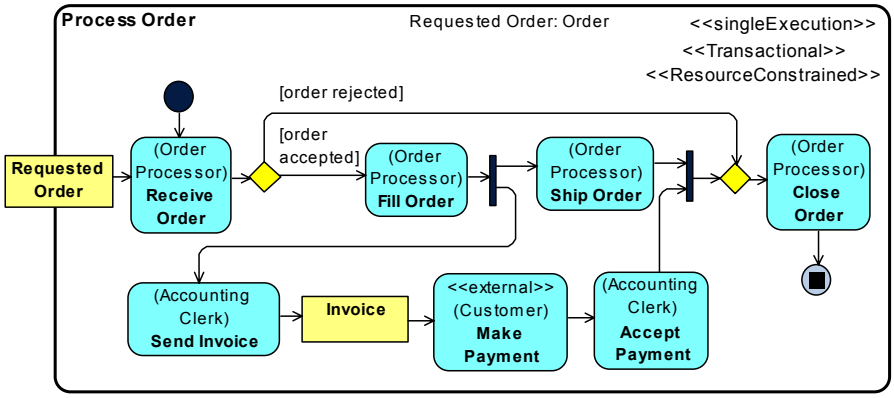

Figure 8. Final version of the example

It should be noted that business models of the accuracy described here typically contain also the execution time (average, maximal,…) for any activity. Since each activity is a class, there are no problems to specify the execution time as an ordinary class attribute. But it could as well be a part of the `Business Process` stereotype, since it

should be used by modeling tools in a uniform way (the formal semantics definition does not require this).

6. CONCLUSIONS

The paper has shown that from the syntax point of view the proposal for UML 2.0 activity diagram is adequate for standard business modeling purposes, but there still are problems with semantics when concurrent subprocesses are considered. A solution to these problems – an extension to the semantics definition has been provided, based on the transaction concept, used in business modeling semantics. The solution is given as a Business Modeling profile, containing the Business Process stereotype for the Activity, with one Boolean attribute isTransactional added. The same profile, with the subprofile ORG Elements included and the Boolean attribute isResourceConstrained added, permits to define the semantics for activity performer specification and resource-constrained execution mode, typically used in business modeling. Thus the proposed profile actually provides the traditional BM semantics for UML activity diagram.

REFERENCES

1. OMG white paper. BUSINESS PROCESSES AND THE OMG. http://www.omg.org/bp-corner/bp-files/The_OMG_BP_Corner_INTRO_Paper_3-2-04.pdf (2004)
2. OMG. Unified Modeling Language: Superstructure. Version 2.0 (Final Adopted Specification). http://www.omg.org/cgi-bin/doc?ptc/2003-08-02 (2003)
3. White S. Process Modeling Notations and Workflow Patterns. The Workflow Handbook 2004. *WfMC, Future Strategies Inc., Lighthouse Point, FL, USA,* 2004
4. BPMI. Business Process Modeling Notation. Working Draft (1.0) August 25, 2003. http://www.bpmi.org/ (2003)
5. Scheer, A.-W.: ARIS Business Process Modeling. 3rd edn. *Springer-Verlag, Berlin Heidelberg New York,* 2000
6. Gartner Inc.: The BPA Market Catches Another Major Updraft. http://www.gartner.com/gc/webletter/idsscheer/issue1/article1.html (2002)
7. Mayer, R., Menzel, C., Painter, M.: IDEF3 Process Description Capture Method Report http://www.idef.com/Downloads/pdf/Idef3_fn.pdf, Knowledge Based Systems Inc (1995)
8. Popkin Software: System Architect. http://www.popkin.com/products/system_architect.htm (2004)
9. Rummler, G., Brache, A.,: Improving Performance : How to Manage the White Space in the Organization Chart. *2nd edn. Jossey-Bass,* 1995
10. Kalnins, A., Barzdins, J. et al: Business Modeling Language GRAPES-BM and Related CASE Tools. *Proceedings of Baltic DB&IS'96, Vol. 2, Institute of Cybernetics, Tallinn*, 1996, pp.3-16
11. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. *PrenticeHall, Englewood Cliffs*, 1981