Towards the Support for Crosscutting Concerns in Activity Diagrams: a Graphical Approach

João Paulo Barros[1,2,3]
[1]Instituto Politécnico de Beja,
Escola Superior de Tecnologia e Gestão,
Área Departamental de Engenharia
Rua Afonso III, n.º 1,
7800-050 Beja, Portugal
jpb@uninova.pt

[2]UNINOVA
2829-516 Monte de Caparica,
Portugal

Luis Gomes[2,3]
[3]Universidade Nova de Lisboa
Fac. de Ciências e Tecnologia,
Dep. de Engenharia Electrotécnica,
Campus da FCT,
2825 Monte de Caparica,
Portugal
lugo@uninova.pt

ABSTRACT

The current proposal for activity diagrams in UML 2.0 allows for very complex graphs, reflecting the numerous interdependencies among the several actions and resources. This paper proposes a graphical composition operation supporting the addition of crosscutting requirements in activity diagrams through node fusion, addition, and subtraction. The operation has a highly readable and intuitive graphical representation and allows simple traceability and coupling. It is illustrated by a set of examples related to the inclusion of mutual exclusion and fair execution across several activity nodes.

Keywords

Crosscutting requirements, activity diagrams, composition, aspect-oriented design

1. INTRODUCTION

When reviewing the extensive UML literature it is easy to notice that, when compared to other UML diagrams, activity diagrams have been given little attention. This is probably due to their status relatively to state diagrams: in UML 1.x [9] activity diagrams are defined as a state diagrams subset. Yet, UML 2.0 [8] clearly separates activity diagrams from state machines. Furthermore, activity diagrams are brought much closer to Petri nets [11]. As stated in [8], "this widens the number of flows that can be modelled". This fact and the additional defined activity node types, clearly increase the activity diagrams modelling capabilities.

We believe the added modelling power, and its independence towards state machines, will certainly give rise to a more frequent use of activity diagrams and to its application in the modelling of a larger number of system types. Also, the different applications and the increase in the number of flows to be modelled will mean more complex models. Finally, the trend towards Model Driven Architecture will certainly contribute to further increase model complexity. To counter model complexity, which easily gives rise to unreadable graphs, this paper proposes the use of an additional composition operation for activity diagrams. It allows model modification in a traceable and additive way. More specifically, the conducted modifications are encapsulated as concerns (in the sense defined by Dijkstra [3]) that can be added to the initial model. The objective is also to maximise model readability even after modifications whose effect is spread across the model. At the coding level, aspect-oriented programming [7] shares these goals. There, the well-modularised modifications with effect across the model are named aspects. More exactly, aspects are defined as well-modularised crosscutting concerns.

A previous paper [2] proposed the use of sub-activities (invoked activities) as aspects. There the aspects "are partial models (diagrams) that, when connected to the model, impose a new behaviour to a set of existent activities". Yet, it was pointed out that the proposed use of sub-activities introduces additional steps in the activity flow. These result from the use of a non-orthogonal construct for composition, namely the asynchronous interface between the existent graph and the added sub-activities, which is based upon the addition of parameter object nodes.

Differently from [2], this position paper proposes the use of a new composition operation (named *activity addition*) for weaving a crosscutting concern into a primary model. This is achieved through fusion of activity nodes in the primary model with activity nodes (of the same class) in the activity diagram modelling the crosscutting concern. Besides the known advantages in the use of an orthogonal composition construct [14], the node fusion also avoids the use of the additional interface nodes, which introduced potential additional steps (depending on the used semantics).

The activity addition operation is inspired by works in the

Petri net area [12, 5], and allows intuitive forms of synchronous and asynchronous communication between activities. It provides support for incremental modification of activity diagrams and is particularly interesting for supporting the specification of crosscutting concerns at the design level, namely by offering a complementary and orthogonal construct to the supported hierarchical one [10].

As stated by Tarr et al. [14], changes should have a low impact and this requires additive rather than invasive changes. For this reason the proposed activity operation forces a structured modifications in the form of node additions and node subtractions.

The paper follows an informal and strongly example-based presentation of activity addition. The examples illustrate the modification of execution policies, namely the addition of arbiters and mutual exclusion related behaviour changes. It is structured as follows. Section 2 gives a brief overview of the new Behavior::Activities used in the presented examples and details the motivations. Section 3 informally defines the activity addition and presents its support as a simple UML profile. Section 4 presents several examples and section 5 concludes and points to some future work.

In this paper we refer to the activities and the respective activity diagrams as specified in the OMG final adopted specification for the "UML 2.0 Superstructure Specification" document [8].

2. MOTIVATION

As already stated, the UML 2.0 specification [8] defines the package *Behavior::Activities* which replaces Activity Diagrams in UML 1.x [9]. The activities in the proposal have a Petri net based semantics and support a more explicit notation regarding hierarchical composition of activities.

As already stated, the use of a Petri net based semantics makes activity modelling more flexible as activities can have multiple token flows at any one time. This deviates from UML 1.x where multiple tokens were typically used only between paired forks and joins. These multiple token flows make a true concurrent semantics more natural for activities. Stevens [13] has even pointed out that contrary to the remaining of UML where interleaving semantics seems more appropriate, a true concurrent semantics seems natural to activity diagrams. A true concurrent semantics is often more adequate to model distributed systems with software and hardware components. The availability of clearly asynchronous elements (object nodes as token deposits) enable an intuitive modelling of asynchronous communication which is often easier to implement in heterogeneous software-hardware systems.

The hierarchical composition of activities now allows the explicit use of input and output places by means of object nodes (token buffers) which act as parameters to sub-activities. The sub-activities are invoked by action nodes. This invocation provides the "dominant decomposition" construct in activity diagrams. Yet, this decomposition construct only allows a limited form of synchronous communication (through control edges) and of asynchronous communication (through parameter object nodes).

Due to the permissiveness and richness of their semantics, an activity diagram can easily become highly complex, not only in its static dimension (its the graphical representation) but also in its dynamic behaviour. Given the necessity of future model changes, the only available decomposition construct (sub-activity invocation), similar to procedural decomposition in textual programming languages, will, in practice, reveal too rigid to incorporated the necessary crosscutting model changes. This will have the known consequence of imposing the scattering of model modification across one, or several, activities with an explicit increase in the complexity of the modified activities. The desired additive nature of model modifications will be at risk as the modifications will impose a new model instead of becoming an added part that can be readily visualised, changed, or even removed.

What is desired is a simple and intuitive way to introduce the desired modification in the least intrusive way. To that end the added modifications must not compromise the visibility of the initial model and must be clearly isolated and identified as an add-on to the initial model: they should be additive.

As activity diagrams are graphs with several different types of nodes, we believe that some type of node fusion, addition, and subtraction is the most intuitive and readable way to specify modifications. Sub-activity invocation provides a *vertical decomposition* construct, similar to procedure decomposition. The static version is sometimes referred as a *refinement*, or a *macro-decomposition*. Therefore we believe an *horizontal composition* mechanism provides the right complement for crosscutting concerns specification. Note that due to the potential complexity and dimension of a single activity graph, even the modelling of a single activity can benefit from an orthogonal modification.

The activity addition operation, defined in the following section, provide a simple additive composition mechanism orthogonal to activity invocation that is able to specify synchronous and asynchronous communication across an activity through the use of any activity node as a potential fusion node (acting as a join node). Consequently, it can be used to specify in a highly flexible way, the addition of model modifications that crosscut several distinct activity parts.

The following section informally defines this operation and shows how it can be supported by a simple UML profile.

3. ACTIVITY DIAGRAM ADDITION

Activity diagram addition is a graph addition binary operation based on node fusion. The following subsection defines the activity diagram addition (from here on simply called *activity addition*) in terms of a three part node partition. It is followed by a subsection presenting a profile-based specification for activity addition.

3.1 Interface nodes, subtraction nodes, and addition nodes

One or more pairs of activity nodes specify the activity addition node fusion. For each pair, each activity node belongs to different activities: one belongs to an activity in the first model (possibly a primary model to be modified) and the

other to the the second model (possibly an activity specifying a crosscutting concern). Formally, the node fusion is specified by a subset of the Cartesian product of the activity nodes set of each activity.

These paired nodes can specify one of two possible roles: they can be *interface nodes*, or they can be *subtraction nodes*. The remaining activity nodes are not paired. They are the ones appearing in the resulting model. We name them *addition nodes*. In summary, we have an activity node partition with three parts:

1. Interface nodes
2. Subtraction nodes
3. Addition nodes

In the resulting model we get one interface node for each interface node pair: the interface node in the first model becomes connected to all the edges connected to the interface node in the second.

For each subtraction node pair, the subtraction nodes in the first and second models are removed together with all the respective edges.

Addition nodes are not affected by the activity addition operation: all addition nodes in both models appear in the resulting model.

Finally, it is important to notice that although activity addition can be used to add one activity to several different activities, it can not originate the interconnection of two previously unconnected activities. This is due to the *ActivityEdge* constraint stating that "The source and target of an edge must be in the same activity as the edge" [8, page 293]. This constraint may seem to weaken the fight against "tyranny of the dominant decomposition". Yet, we do not intent to break the activity encapsulation, as these would bring additional complexity to the model. The constraint still allows the application of a crosscutting concern to several different activities. The crosscutting should be seen as token flow crosscut inside a given activity or sets of independent activities.

Note also that we restrict node fusion to its graphical part, more specifically we do not handle node operations and attributes when the respective nodes are fused. This implies we should view the presented examples as non-dependent on non-graphical specifications.

Next we show how to use a simple UML profile to specify activity addition, and classify activity addition in *pure activity addition* and *activity subtraction*.

3.2 UML support for activity addition

An important question is how to specify in a UML diagram the presented node fusion. As the UML 2.0 specification has no general support for node fusion, we have to rely on one (or two) of the available extension mechanism: *Profiles* and *MOF*-based extensions[1]. The profiles specification clearly states that "it is not possible to take away any of the constraints that apply to a metamodel such as UML using a profile (...)". Through MOF-based extensions one can to modify the UML meta-model. Profiles only allow for meta-model extensions. Naturally, one should try to avoid modifications to the metamodel if possible and this is the case for the specification of the presented node partition: we only use a Profile for providing the necessary data specifying interface node pairs and subtraction node pairs. All unspecified nodes default to addition nodes.

The Profiles specification states that "the profiles mechanism does not allow for modifying existing metamodels" [8, page 569]. Consequently, we can not add some type of node removal capability as a way to specify the fusion of two interface nodes or operation nodes. Yet, through a profile we can specify the data necessary to transform one model into another, that is, to compose (*weave*) the crosscutting concern into the model. In fact, model transformation is explicitly suggested as a possible application for profiles: "Add information that can be used when transforming a model to another model (...)" [8, page 569]. To that end we define a profile named *ActivityAddition* containing two stereotypes applicable to all activity nodes (see Fig. 1).

The two stereotypes are named, respectively, *InterfaceNode*, and *SubtractionNode*. Both extend the *ActivityNode* class with an attribute specifying the node that should be fused with it. In both cases we impose a constraint restricting the fusion of activity nodes to nodes of the same class: e.g. a fork node can only be fused (as an interface node or subtraction node) with another fork node.

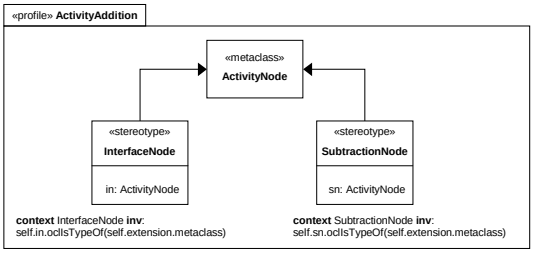

Figure 1: *ActivityAddition* profile.

It is important to notice that a third stereotype for addition nodes can be defined. Yet, the classification as an addition node is the default for all remaining nodes (that are neither interface nodes nor subtraction nodes), we decided to use this fact as a way to simplify the diagram notation. This means no addition node is stereotyped.

Fig. 2 exemplifies activity addition by showing how the activity *Process Order* from [8, Fig. 203 on page 290] can be constructed by the addition of two initially unrelated activities (*Order Handling* and *Invoice Handling*). Note the use of node naming for the fork (a) and join (b) nodes in the Order Handling activity, and the reference to those names by the attributes *in* in the stereotyped join and fork in activity Invoice Handling. Interestingly, the use of join and fork nodes with a single incoming and a single outgoing edge is

[1]MOF means Meta Object Facility

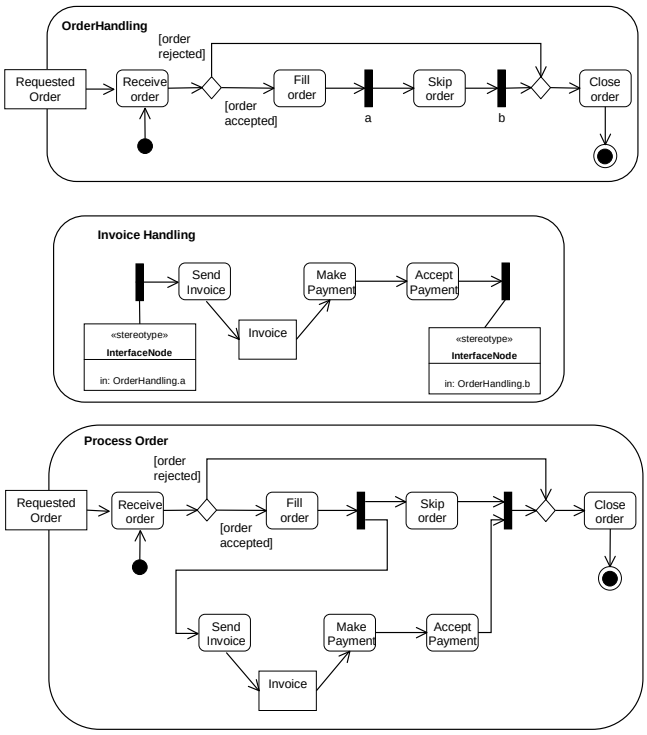

Figure 2: Activity addition and the respective result.

useful for activity addition specification. In fact, their use provides the desired support for activity fusion by providing an additional *fusion point* that is also a *join point* (see below). Sometimes simple edges should be transformed into a fork or join with unique incoming and outgoing edges primary to their fusion. The same is true for *DecisionNode*s and *MergeNode*s. The examples to be presented also illustrate this prior transformation from edges to fork or join nodes, with a single incoming and a single outgoing edge. We will call these *transparent fork/join nodes*.

3.3 Pure addition and subtraction

Activity modification through addition becomes more powerful if it is possible to remove nodes. This is provided by subtraction nodes. Subtraction nodes (see Fig. 1) can be seen as negative elements in node fusion. This means they specify which nodes should be removed from the initial model. In the context of a control version system for UML models, this notion of negative elements also appears (as "negative model elements") in the paper by Alanen and Porres [1].

Fig. 3, shows the effect of a subtraction node. The specification of the action node as a subtraction node associated to node c in activity X results in activity Z.

When all nodes in the second activity diagram are classified as interface nodes or subtraction nodes, we talk about *net subtraction*. Subtraction nodes can also be seen as fusion nodes, which, after fusion, are removed together with all the connected edges.

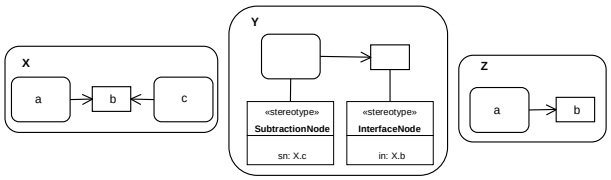

Figure 3: General activity addition (X+Y=Z).

If there are no subtraction node stereotypes, we talk about *pure net addition*, to differentiate from *general net addition* (see Fig. 3) where addition nodes appear together with subtraction nodes.

As a rule, pure net addition, and net subtraction should be preferred over general net addition as they provide more readable a traceable model modifications.

Typically, activity subtraction provides a clean away to undo previous modifications specified by pure net addition. On the other hand, general net addition allows more complex modifications to be specified. Unsurprisingly, another general net addition (a dual one) will be necessary to undo those modifications.

The following sub-sections present three examples of activity diagram modification through the use of pure activity addition. In all cases activity subtraction can be used to undo the effect of a previous node addition.

4. EXAMPLES

This section uses the examples in [2] to illustrate possible uses of activity addition for the modification of existing models. We start by presenting the application of a mutual exclusion to existent and independent token flows; after, we present the application of simple round-robin arbiter forcing a fair execution of a set of token flows; next we "merge" mutual exclusion and fair execution, and the last example sets a preferred choice in a set of possible token flows. In all cases, activity subtraction allows the *rewind* of the model to its initial state, that is, the undo of the modifications resulting from the activity addition.

4.1 Mutual exclusion

As a first example of a crosscutting concern in activity diagrams, we consider the case where we want to impose a mutual exclusion between two, or more, independent activities. On the left of Fig. 4 we have two control flows: one between a and b and another between c and d. On the right we have the same diagram but with the four edges replaced by transparent forks/join nodes. These will be useful for activity addition. We want activities b and d to be mutually exclusive. To that end we will add a locking mechanism: each action must get a token to be executed; after it releases the token. Activity *L1* in Fig. 5 specified this simple behaviour. Activity *L2* is another instance of the same activity but "stereotyped" by the specification of the object node *Lock* as an interface node. *L1* and *L2* can then be added resulting in activity *L1+L2* (see it already stereotyped in Fig. 6). We can now apply the interface node stereotype to add activity *L1+L2* to activity A1' (see Fig. 6). We this we get the modified model in Fig. 7.

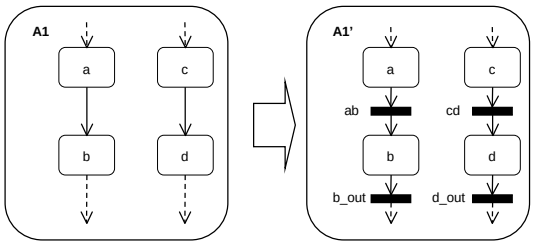

Figure 4: Introducing transparent fork/joins in two independent token flows.

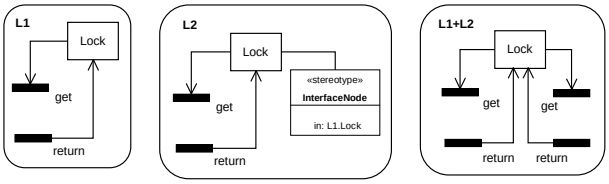

Figure 5: Composing a crosscutting concern from two instances of an activity.

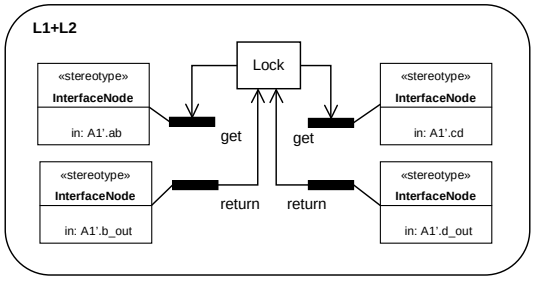

Figure 6: A stereotyped activity for the crosscutting concern on the right of Fig. 5.

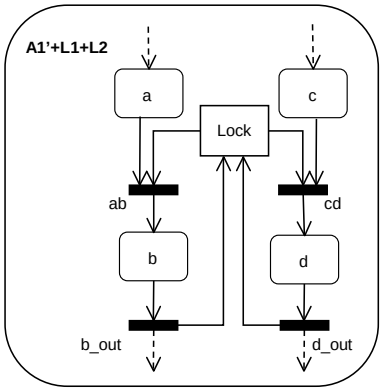

Figure 7: The activity in Fig. 4 modified by the activity in Fig. 6.

The modification introduced by the model in Fig. 6 can be readily undone by the respective dual model (see Fig. 8). Notice that the dual uses the same interface nodes (the "glue" points) and reverses the roles of the remaining node: the Lock object node that was, implicitly, an addition node in Fig. 6 is stereotyped as a subtraction node in the dual model. As laready defined, in this case, where all nodes in the second model are either interface nodes or subtraction nodes, we talk about activity subtraction. Although this dual operation can always be applied we will omit it in the following examples so as to avoid unnecessary burden in the presentation.

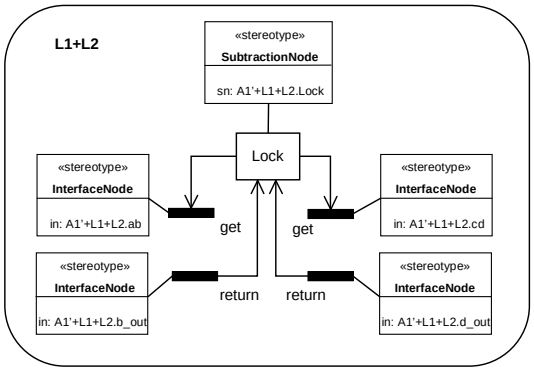

Figure 8: The dual of Fig. 6 model.

Note that the self-addition of the crosscutting concern *L1* in Fig. 5, effectively provides a way to generalise the application of the crosscutting concern prior to its application to the model to be modified. This use of activity addition is also used in the following examples.

4.2 Fair execution

Conflict resolution is an important issue when modelling critical systems. Sometimes conflicts are undesirable because they model non-determinism and this may lead to unpredictable behaviour; other times conflicts are necessary, e.g. mutual exclusion semaphores or resource sharing. The activities semantics in the UML 2.0 specification does not provide an answer to conflict resolution: conflict resolution requires specific user intervention. These interventions can correspond to known strategies that can be applied in form of pre-existent models. For example, the decision node is a potential source of conflicts as it can easily originate non-determinism if the outgoing edges guards do not guarantee mutually exclusive token flows. The left of Fig. 9 shows an example of a decision node with three possible outgoing flows. The right side shows the same diagram with the added transparent fork/join nodes).

A fair selection of the several possible token flows outgoing from a decision node constitutes an example where a known model specifying a crosscutting concern can be applied. This can be specified by a token-player where each token flow passes its turn to another token flow. Activity *Transfer* in Fig. 10 specifies this token transfer in an elementary way. In the same figure, activity *Init* provides the support for specifying the initial token position. It should be clear that activity *3Transfer* can be readily obtained from the composition of three instances of activity *Transfer* with one instance of activity *Init*. When added to the initial activity *A'*, the resulting activity acts as an arbiter: Fig. 11 shows the stereotyped *3Transfer* activity and the result of its addition to activity *A'*.

If we want to avoid the introduction of transparent fork/join nodes as a prior condition to activity addition, one can usu-

ally find some other interface nodes (fusion points) allowing the same final result. Fig. 12 exemplifies this fact with an an alternative 3Transfer activity that provides the same final result when added to the initial activity *A* in Fig. 9. As already mentioned, all interface nodes must be seen as graphical elements with irrelevant additional characteristics (as attributes, operations, or others), namely the actions in *3Transfer* do not affect the existent actions (*a*, *b*, and *c*):

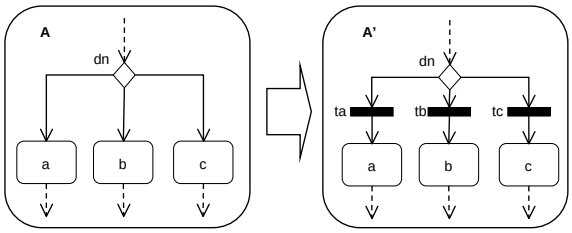

Figure 9: Primary model and the same model with with transparent fork/joins.

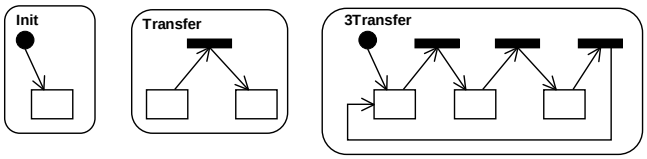

Figure 10: Activity *Init* and activity *Transfer*. *3Transfer* activity resulting from the activity addition of three *Transfer* instances and one *Init* instance.

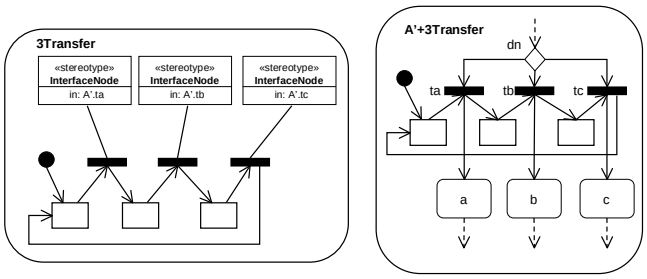

Figure 11: Stereotyped 3Transfer activity and the result from its addition to the model on the right of Fig. 9.

As already noticed in [2] the outgoing edge guards must evaluate to true so as to avoid deadlocks, and the fairness ratios can easily be changed by the addition of join/fork pairs associated to the same output action.

4.3 Fair mutual exclusion

If we bring together the characteristics of mutual exclusion and fair execution, we get a fair choice among mutual exclusive processes. The token is transferred between process locks. Fig. 13 shows this lock transfer on the left (activity *LT*). We opted for a separated activity modelling the lock initialisation (activity Init on the right of Fig. 13).

Similarly to the previous examples, we can compose, by activity addition, several instances of activity *LT*. Fig. 14 illustrates this addition: one *LT* instance is stereotyped with

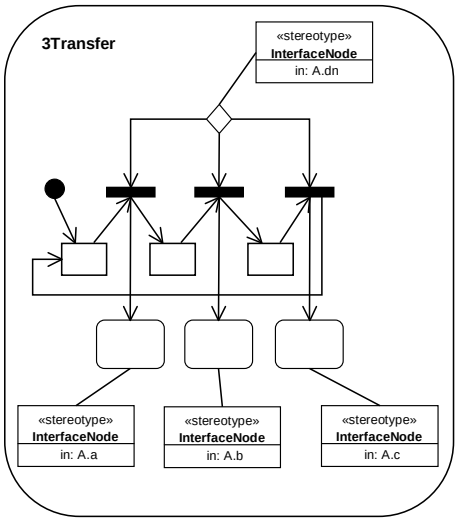

Figure 12: An alternative stereotyped crosscutting requirement to be added to the initial model *A* in Fig. 9 (without transparent join/fork pairs).

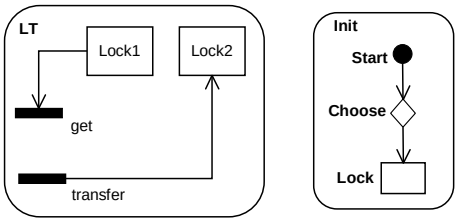

Figure 13: Activity *LT* and activity *Init*, used for composing a fair mutual exclusive crosscutting concern.

two interface nodes. An *Init* activity instance (*Init1*) is them added to the resulting activity diagram (Fig. 15). The resulting activity diagram (*LT1+LT2+Init*) is shown on the left of Fig. 16. It already specifies a crosscutting concern for an alternate mutual exclusion with deterministic initialisation (*Lock1* always receives the initial token).

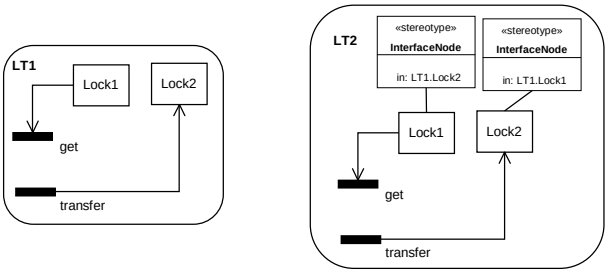

Figure 14: Two *LT* instances to be added.

We can further add another *Init* instance to obtain a nondeterministic choice of *Lock*s: adding Init2 (right of Fig. 16) we get the activity in Fig. 17 without the stereotypes. The shown stereotyped LT1+LT2+Init1+Init2 activity can finally be added to the A1' activity in Fig. 4. We get activity *A1'+LT1+LT2+Init1+Init2* (see Fig. 18).

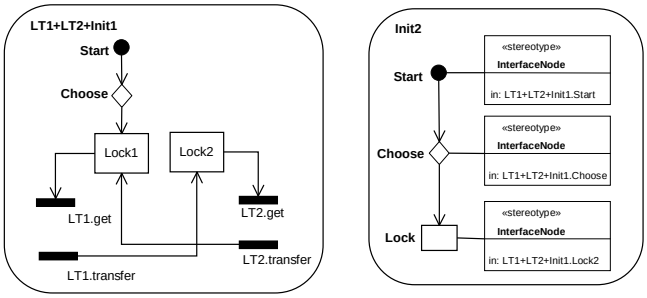

Figure 15: Result of the activity addition in Fig. 13 (on the left) and the stereotyped *Init* instance to be added (on the right).

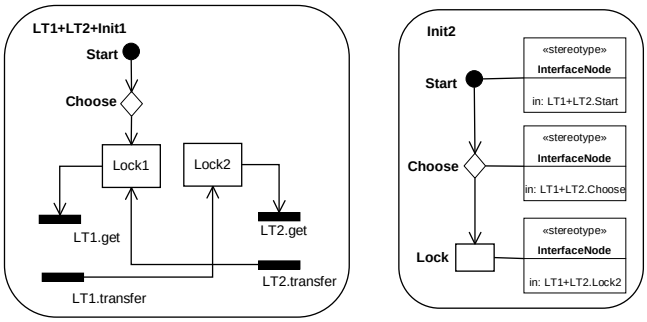

Figure 16: Result of the activity addition in Fig. 15 (on the left) and the stereotyped *Init* instance to be added (on the right).

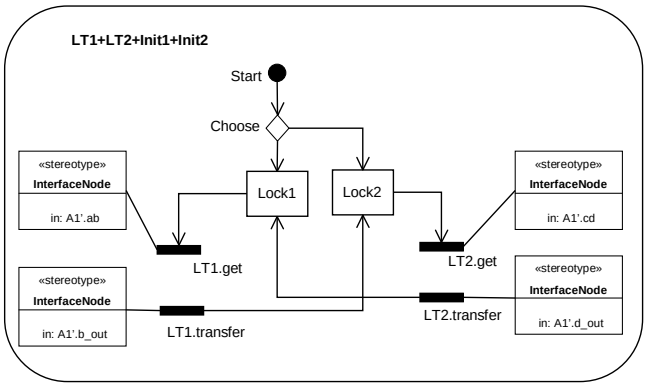

Figure 17: Result of the activity addition in Fig. 16.

5. CONCLUSIONS AND FUTURE WORK

The UML 2.0 activity diagrams have an much richer semantics. This fact will certainly give rise to a much more widespread use of activity diagrams, namely in other areas, besides workflow systems. Areas such as critical systems with distributed hardware and software components now seem suitable to be modelled by activity diagrams as they are able to provide an alternative to state machines. In this setting, it seems easy to foresee that the support for crosscutting requirements will become highly desirable. The proposed activity composition by graph addition is a contribution in this direction. Node fusion provides a readable and intuitive composition mechanism allowing a non-

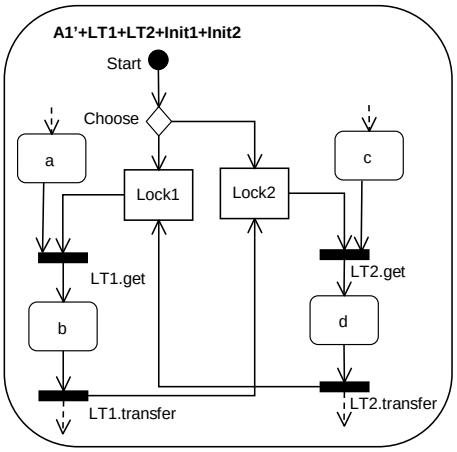

Figure 18: Modified model resulting from activity addition of activity in Fig. 17 to activity A1' in Fig. 4.

invasive addition of crosscutting concerns to existent models, with minimal coupling and simple traceability. Interestingly, the support for crosscutting concerns here presented does not require any metamodel extension as it relies exclusively on profiles based extensions. Yet, adequate tool support is clearly necessary to permit an automatic generation of the resulting models.

The proposed node fusion based on a three part node partition (interface nodes, subtraction nodes, and addition nodes) seems readily applicable to other behaviour diagrams, namely to UML state machines. Presently, we can only guess that node fusion is probably also useful for the specification of crosscutting requirements in non-behaviour diagrams.

In *AspectJ* terms [6], the activity nodes are all potential *join points* of the primary model activity diagram; and the fusion pairs can be seen as particular cases of *pointcuts*. In that sense other more elaborate pointcuts can be useful. For example, the use of pairs containing one set of nodes in the primary model and one node in the crosscutting concern model. This would allow the specification of several fusions, between several nodes in the primary model with a single node in the crosscutting concern model. In this sense the definition of more elaborate fusion based pointcuts will be part of our future work.

The fusion of non-graphical activity node specifications, namely node operations and attributes, also needs to be defined. The presented proposal leaves this open. Finally, tool support in the form of XMI transformations is probably the best way to rapidly put the proposed activity operation to practice.

6. REFERENCES

[1] M. Alanen and I. Porres. Difference and union of models. In *UML 2002 - The Unified Modeling Language 6th International Conference(UML'2003)*, oct 2003. (to appear).

[2] J. P. Barros and L. Gomes. Activities as behaviour aspects. In M. Kandé, O. Aldawud, G. Booch, and

B. Harrison, editors, *Workshop on Aspect-Oriented Modeling with UML*, sep 2002. Satellite workshop of the UML 2002 - The Unified Modeling Language 5th International Conference(UML'2002).

[3] E. W. Dijkstra. On the role of scientific thought. published as [4], Aug. 1974.

[4] E. W. Dijkstra. On the role of scientific thought. In *Selected Writings on Computing: A Personal Perspective*, pages 60–66. Springer-Verlag, 1982.

[5] L. Gomes and A. Steiger-Garção. Towards the implementation of conflict resolution on a non autonomous high-level petri net model. In *Workshop Manufacturing and Petri Nets, Osaka, Japan*, jun 1996. Satellite workshop of the 17^{th} International Conference on Application and Theory of Petri Nets (ICATPN'96).

[6] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and William G. Griswold. An overview of aspectj. In J. L. Knudsen, editor, 15^{th} *European Conference on Object-Oriented Programming*, volume 2072 of *LNCS*, pages 327–353, Berlin, Heidelberg, and New York, 2001. Springer Verlag.

[7] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, 11^{th} *European Conference on Object-Oriented Programming*, volume 1241 of *LNCS*, pages 220–242, Berlin, Heidelberg, and New York, 1997. Springer Verlag.

[8] OMG. Uml 2.0 superstructure specification. `http://www.omg.org/cgi-bin/doc?ptc/03-08-02`, August 2003. version 2.0. Final Adopted Specification. OMG Adopted Specification ptc/03-08-02.

[9] O. M. G. (OMG). Unified modeling language specification, version 1.5. `http://www.omg.org/cgi-bin/doc?formal/03-03-01.pdf`, mar 2003.

[10] H. Ossher and P. Tarr. Using multidimensional separation of concerns to (re)shape evolving software. *Communications of the ACM*, 44(10):43–50, 2001.

[11] W. Reisig. *Petri nets: an introduction*. Springer-Verlag New York, Inc., 1985.

[12] E. Smith. Arbiter behaviour and petri nets. *Fachberichte Informatik, Universität Koblenz-Landau, 2/93*, 1993.

[13] P. Stevens. Uml and concurrency. Available at `http://www.dcs.ed.ac.uk/home/pxs/asmtalk.pdf`, March 2003. Invited talk at 10th International Workshop on Abstract State Machines (ASM 2003).

[14] P. Tarr, H. Ossher, W. Harrison, and J. Stanley M. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *Proceedings of the 21st international conference on Software engineering*, pages 107–119. IEEE Computer Society Press, 1999.