Integrating UML Activity Diagrams with Temporal Logic Expressions

João Araújo and Ana Moreira

Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Quinta da Torre, Caparica, PORTUGAL
Tel: +351-21-294 8536; Fax: +351-21-294 8541
E-mail: {ja, amm}@di.fct.unl.pt

Abstract. UML is a standard modelling language that is able to specify a wide range of object-oriented concepts. However, the diagrams it offers are many times accused of lack of rigour to specify precisely some critical requirements and therefore it is often needed to complement the semantics of the UML diagrams using OCL or any other formal language. In the case of activity diagrams (used here to describe use cases), OCL is not the most appropriate formal language, as it does not represent temporal aspects directly. Our aim is to complement the well-accepted simplicity of activity diagrams with a temporal logic specification to give a more precise semantics to the final model. This specification can be further used to validate requirements against the stakeholders using animation techniques.

Keywords: Unified Modeling Language (UML), Requirements Engineering, Temporal Logic.

1 Introduction

UML (Unified Modeling Language) [OMG 2005] has a very rich notation for modelling both structural and behavioural aspects of a system. The semantics associated to its main constructors, concepts and techniques is defined by means of a metamodel. When additional formal constraints or data is needed in a given model, most researchers use OCL [Warmer and Kleppe 2003]. OCL can then be used to augment the expressive power of the structural model. However, to specify UML behavioural models, OCL does not provide appropriate constructs.

Building a formal specification of a system requires a rigorous set of rules to transform a set of informal requirements into a formal specification, bridging the gap between the two representations. Formalisation is useful so that we can identify ambiguities, inconsistencies and incompleteness earlier in the software development process.

Our goal is to describe the integration of object models and formal specification languages. In particular, we will integrate UML activity diagrams with temporal

logic. We have chosen temporal logic because it provides an effective way to represent formally the temporal aspects of a system behaviour. Moreover, temporal logic has been used with success in software development at programming level [Moszskowski 1986, Kröger 1987]. Therefore, integrating activity diagrams with temporal logic has the potential of allowing the validation of scenarios described by those diagrams earlier on during software development.

As mentioned above, OCL is used in the context of UML by many researchers. However, it was not adopted here due to the temporal intrinsic nature of the technique in hand (i.e. activity diagrams). OCL is not the most appropriate formal language to specify temporal constraints. On the other hand, temporal logic was created exactly for that purpose. In summary, the advantage of our proposal is two fold:

- to add rigour to the behavioural UML models, in particular to activity diagrams;
- to use the resulting formal specifications to generate prototypes in order to validate requirements together with the stakeholders. Therefore, animation techniques can be used here, although this is out of the scope of this paper.

This paper is organised as follows. Section 2 revisits activity diagrams as incorporated within UML. Section 3 introduces an example that will be used to illustrate the approach. Section 4 shows how to specify formally activity diagrams and applies the results to our example. Section 5 presents some related work. Finally, Section 6 draws our conclusions and points out directions for future work.

2 Activity diagrams: an overview

Activity diagrams are frequently used to describe use cases or their scenarios. Use cases, as proposed by [Jacobson 1992], describe functional requirements of a system, helping to identify the complete set of user requirements. A use case describes a generic transaction, normally involving several objects and messages. Industrial software developers are easily seduced by the simplicity and potentiality of use cases; they claim that use cases are an interesting and easily understood technique for capturing requirements.

Use cases are described by scenarios that can be more rigorously represented by activity diagrams. An activity diagram consists of action states, activity states, transitions, object flows, branching, forks and joins, and swimlanes.

Action states are system states that represent the execution of an action (e.g. create or destroy an object, send a signal to an object). They are atomic and, therefore, they cannot be decomposed, or interrupted. An action state lasts an insignificant amount of time. Activity states can be decomposed, i.e., one activity state can be described by another activity diagram. Therefore they are non-atomic and can be interrupted. The notation is the same as the action state.

Transitions occur when an action or activity ends; the flow of control passes immediately to the next action or activity. Branching is used to represent alternative transitions, and is based on a boolean expression (a guard). A branch consists of one

incoming and two or more outgoing transitions. The guards must cover all the possibilities, but they cannot overlap.

A join represents the synchronization of two or more concurrent flows of control and has two or more incoming transitions and only one outgoing transition. The fork represents the splitting of one control flow in two or more control flows and has one incoming transition and two or more outgoing transitions.

3 Example

The model components of an activity diagram described in the previous section will be discussed and formalised using a simplified version of a system to deliver a product to a customer. The requirements of such a subsystem are as follows:

> "A customer should be able to register in the system. This registration results in opening an account for him that will be administrated by an Accounting System. Afterwards, s/he can buy a product whose shipping is realized by a Shipping Company. The Accounting System captures the order. The customer can also cancel an order, or return the product. As a consequence, these services have the effect of updating the customer's account."

Analysing the requirements described above, we can build the use case diagram depicted in Figure 1. We can identify three actors: *Customer, Accounting System*, and *Shipping Company*. The use cases identified are *Register Customer, Process Order, Cancel Order,* and *Return Product*.

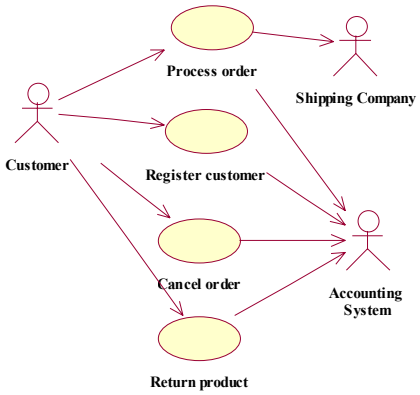

Fig. 1. Use case diagram of order processing system

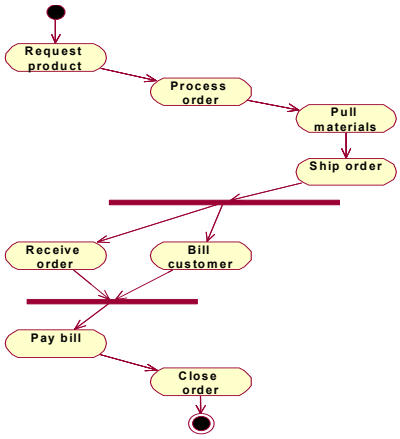

Fig. 2. Activity diagram for the use case Process Order

The activity diagram for the use case Process Order is shown in Figure 2. When the customer requests a product the order processing is started and the product is pulled and shipped. The customer should receive the order and the bill. Once paid, the order is closed.

4 Using Temporal Logic to augment activity diagrams semantics

Activity diagrams are a powerful technique to describe requirements in use cases. However, this is not enough to guarantee that the requirements do not contain errors, ambiguities, omissions and inconsistencies. These drawbacks can only be identified and corrected early in the development process if formal description techniques are used. The aim here is to obtain a formal specification from activity diagrams. This will help us reasoning about the information specified in the activity diagram to identify ambiguities and incorrectness. The rules to generate a temporal logic specification from activity diagrams are based on safety, guarantee and response properties of programs that can be specified by temporal logic formulas [Manna and Pnuelli 1992] and the temporal logic operators $\Box$ (always) and $\Diamond$ (eventually). Such properties are defined as follows:

1. *Safety Property*: can be specified by a safety formula. A safety formula is any formula that is equivalent to a canonical safety formula $\Box p$ (where p always holds). Usually, safety formulas represent invariance of some state property over all the computations.
2. *Guarantee Properties*: can be specified by a guarantee formula. A guarantee formula is equivalent to a canonical formula of the type $\Diamond p$. This states that p eventually happens at least once in the future.
3. *Response Properties*: can be specified by a response formula. A response formula is equivalent to a canonical formula of the type $\Box\Diamond p$. This states that every stimulus has a response. An alternative formula is $\Box(p \rightarrow \Diamond q)$, which states that every p is followed by a q, that is, q is a guaranteed response to p.

These properties can be classified into safety and progress (or liveness). A safety property states that a requirement must always be satisfied in a computation. Progress properties can be either guarantee or response. The progress properties specify a requirement that should eventually be fulfilled. Therefore, they are associated with progress towards the fulfilment of the requirement.

Temporal logic can specify progress issues by showing how the various activities interact, for example, when specifying the priority, or the order in which activities may happen Activity diagrams show the flow of control from activity to activity, which can naturally be expressed by temporal logic, where activity and action states are our properties.

Based on the three general properties above, we can define 5 rules that state how an activity diagram is transformed into a temporal logic expression:

1. A simple transition, in an activity diagram, from an action or an activity state α_i to another action or activity state α_{i+1}, can be formalised as $(\alpha_i \rightarrow \alpha_{i+1})$.
2. In the case of a sequential transition, we have:
   - if there is only one action or activity state, this can be mapped into the canonical formula $\Box\Diamond\alpha_i$, where i = 1; otherwise,
   - if there is a sequence of states, the general response form is $\Box(\alpha_i \rightarrow \Diamond\beta)$, where α_i represents the first action or activity state and β the rest of the sequence. β has two forms:

- α_j with $1 < j \leq n$, to deal with the last action or activity state, and
- $\alpha_j \rightarrow \Diamond (\alpha_{j+1} \rightarrow \ldots \Diamond (\alpha_{n-1} \rightarrow \Diamond \alpha_n) \ldots)$ where $1 < j \leq n$.

3. In case of branching, we have associated conditions (*condition* $_k$, *condition* $_{k+1}$, ..., *condition*$_{k+n}$) to the transitions. Therefore we have the expression:
   - $(condition_k \wedge \alpha_i \rightarrow \Diamond\alpha_{i+1}) \vee condition_{k+1} \wedge \alpha_i \rightarrow \Diamond\alpha_{i+2,} \vee \ldots condition_{k+n} \wedge \alpha_i \rightarrow \Diamond\alpha_{k+n+1}$ where $1 \leq i \leq n$ and $1 \leq k \leq n$.
4. When we have a fork:
   - if there is a transition from an action or activity state α_i to a group of concurrent action or activity states γ_k (represented by a fork), this can be mapped to the formula $\alpha_i \rightarrow \Diamond\gamma_k$, where $\gamma_k = \wedge^n_{p=i+1} \alpha_p$, where $\wedge$ is the conjunction of all the activity states α_i.
5. When we have a join:
   - if we have a group of concurrent action or activity states γ_k, being joined and transited to an action or activity state α_j the general form is $\gamma_k \rightarrow \Diamond \alpha_j$, where γ_k is as before (step 4) , and $j > p$, $i + 1 < p < n$.

Following the mappings just discussed, it is not difficult to transform the *Process Order* activity diagram in Figure 2 into the temporal logic expression in Figure 3.

$\Box$ *(requestProduct* $\rightarrow$
$\Diamond$ *(processOrder* $\rightarrow$
$\Diamond$ *(pullMaterials* $\rightarrow$
$\Diamond$ *(shipOrder* $\rightarrow$
$\Diamond$ *((receiveOrder* $\wedge$ *billCustomer)* $\rightarrow$
$\Diamond$ *(payBill* $\rightarrow$
$\Diamond$ *(closeOrder)))))))*

Fig. 3. Temporal logic expression for the activity diagram Process Order

The resulting temporal logic expressions could be used to validate requirements against the users by means of animation techniques. There is some work on programming languages based on the execution of temporal logic such as Tokio [Fujita et al. 1986], METATEM [Barringer et al. 1989] and FTLL [Duan and Koutny 2004] that could be used to implement the formal specifications obtained using our transformation algorithm.

As a final note, we would like to emphasize that our approach could be used to build incrementally a complete formal specification of the system, or at the least the critical parts of the system, where all the temporal logic expressions should be composed accordingly. In such a scenario, validation would be realized incrementally.

5 Related work

Producing a formal specification from object-oriented models is not new. During the nineties, many researchers have written on how to integrate object-oriented methods with formal description techniques. For example, Moreira and Clark developed

ROOA (Rigorous Object-Oriented Analysis) [Clark and Moreira 1999, Moreira and Clark 1996] to build a formal and executable object-oriented specification from informal requirements using SDL [Z.100 1994] or LOTOS [ISO 1998]. Araújo and Sawyer presents an approach (Metamorphosis) [Araújo and Sawyer 1998] to combine an object-oriented model with Object-Z [Duke et al. 1991]. However, none of them considers formalising UML models [OMG 2005].

The precise UML (pUML) group was created with the aim of explicitly contributing to the formalization of the UML language. This group undertook collaborative work to use formal techniques to explore and define appropriate semantic foundations for object-oriented concepts and UML notations. A list of work produced under this affiliation can be found on http://www.cs.york.ac.uk/puml/.

France discusses the formalisation of the UML static requirements modelling concepts [France 1999]. Overgaard gives a formal definition of the collaboration construct in the UML [Overgaard 1999]. Knapp presents a formal semantics for UML interactions [Knapp 1999]. Additionally, Kim and Carrington shows the formalisation of the UML class diagram using Object-Z [Kim and Carrington 1999]. Cabot et al. propose an extension to UML to define a set of temporal features of entity and relationship types, and provide a notation to refer to any past state of the information base [Cabot et al. 2003], but it does not explain clearly how these could be used for activity diagrams. Ziemann and Gogolla proposes an extension of OCL with temporal logic, but this extension is only applied to class diagrams [Ziemann and Gogolla 2003]. Giese and Heldal investigate the relationship between the informal pre- and post-conditions of use cases and the formal OCL pre- and postconditions of operations in the class diagram [Giese and Heldal 2004]. However, temporal aspects are not considered here.

In some of our previous work [Araújo and Moreira 2000, Moreira and Araújo 2000], we define mappings of use cases, sequence diagrams and collaboration diagrams into Object-Z class schemas. Temporal aspects are captured in Object-Z's history invariants.

A different kind of work on formalizing activity diagrams is presented in [Eshuis and Wieringa 2001]. Here, the authors define a formal execution semantics for UML activity diagrams whose goal is to support execution of workflow models and analysis of the functional requirements that these models satisfy. Also, in [Baresi and Pezzè 2001] activity diagrams and other UML specifications are formalized with high-level Petri nets through the definition of translation rules.

In spite of all this work, there is still a lot to be said about formal specifications. Currently, lots of work on formal specification have been done as part of the MDA agenda. Our work is related to this as our formalizations are, in fact, model transformations. Therefore, our results can be useful to the MDA community.

7 Conclusions and future work

The process described in this paper provides a set of rules to transform activity diagrams into a temporal logic expression. The integration of the two approaches is synergetic because the sum of the advantages of these approaches is greater than if they are considered in isolation. Some of the advantages identified are:

(i)formalisation at early stages to achieve more precise and better quality systems is encouraged; (ii) normalisation of different notations into one precise mathematical notation; (iii) a deep reasoning about the system is promoted, as the language has a mathematical semantics.

Nevertheless, more work must be done. In particular, we need to extend our approach to handle object flows, when it is desirable to show objects that are produced by certain activities. Also, we need to conduct real case studies to validate our approach. Moreover, to increase the usability of our approach, it would be interesting to investigate how Tokio, METATEM and FTLL languages could be effectively integrated to our approach.

Another task needing investigation is the expected evolution of the resulting specification into a process that builds a specification centred on objects. To improve the process, reverse engineering should be defined, i.e., from temporal logic formulas we should obtain the activity diagrams automatically. This is useful to promote modifiability and traceability. Furthermore, since activity diagrams are closely related to Petri nets, to the extent that in UML 2.0 [OMG 2005] they were enriched with Petri nets elements, we intend to investigate how our approach can be extended to contemplate Petri nets features,

We do not believe that integrated methods alone will make the use of formal specification widespread. Other pragmatic issues must be taken into consideration such as training and a change in the culture of the organisation. However we are optimistic that methods like the one described here can contribute to the effective incorporation of formal specification by industry.

References

[Araújo and Moreira 2000] Araújo, J., Moreira, A., "Specifying the Behaviour of UML Collaborations Using Object-Z", *Americas Conference on Information Systems (AMCIS), Object-Oriented Software Development Mini-Track*, California, USA, August 2000.

[Araújo and Sawyer 1998] Araújo, J., Sawyer, P., "Integrating Object-Oriented Analysis and Formal Specification", *Journal of Brazilian Computer Society* (special edition on software engineering), Campinas, Brazil, July 1998.

[Baresi and Pezzè 2001] Baresi, L., Pezzè, M., "On Formalizing UML with High-Level Petri Nets", *Concurrent Object-Oriented Programming and Petri Nets: Advances in Petri Nets,* eds. G.A. Agha, F. De Cindio, and G. Rozenberg, Lecture Notes in Computer Science, Vol. 2001, 2001.

[Barringer et al. 1989] Barringer, H., Fisher, M., Gabbay, D., Gough, G., Owens, R., "MetateM: A Frameweork for Programming in Temporal Logic", *Workshop on Stepwise Refinement of Distributed Systems*, Netherlands, Lecture Notes in Computer Science, Vol. 430, June 1989.

[Cabot et al. 2003] Cabot, J., Olivé, A., Teniente, E., "Representing Temporal Information in UML", *UML 2003*, Lecture Notes in Computer Science, Vol. 2863, Springer-Verlag, October 1999, pp. 44-59.

[Clark and Moreira 1999] Clark, R., Moreira, A., "SDL in Rigours Object-Oriented Analysis", *Formal Methods for Open Object-Based Distributed Systems*, eds. P. Ciancarini, A. Fantechi, R. Gorrieri, Kluwer Academic Press, 1999.

[Duan and Koutny 2004] Duan, Z., Koutny, M., "A framed temporal logic programming language", *Journal of Computer Science and Technology*, Vol. 19 (3), 2004, pp.341-351.

[Duke et al. 1991] Duke, D., King, P., Rose, G., Smith, G., "The Object-Z Specification Language Version 1", *Technical Report 91-1*, Department of Computer Science, University of Queensland, May 1991.

[France 1999] France, R., "A Problem-Oriented Analysis of Basic UML Static Modeling Concepts", *OOPSLA'99*, ACM Sigplan Notices, Vol. 34(10), October 1999, pp 57-69.

[Eshuis and Wieringa 2001] Eshuis R., Wieringa, R., "A formal semantics for UML activity diagrams – Formalising workflow models", *Technical Report CTIT-01-04*, University of Twente, Holland, 2001.

[Fujita et al. 1986] Fujita , M., Kono, S., Tanaka, H., Moto-Oka, T., "Tokio: Logic Programming Language Based on Temporal Logic and its Compilation to Prolog", *Third International Conference on Logic Programming*, July 1986, pp.695-709.

[Giese and Heldal 2004] Giese, M., Heldal, R., "From Informal to Formal Specifications in UML", *UML 2004*, Lecture Notes in Computer Science, Vol. 3273, October 2004.

[ISO 1998] ISO: Information Processing Systems – Open Systems Interconnection, "LOTOS: A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", *International Standard 8807. ISO*, 1988.

[Jacobson 1992] Jacobson, I., *Object-Oriented Software Engineering – a Use Case Driven Approach*, Addison-Wesley, 1992.

[Kim and Carrington 1999] Kim, S., Carrington, D., "Formalizing the UML Class Diagram Using Object-Z", *UML 1999*, Lecture Notes in Computer Science, Vol. 1723, Springer-Verlag, October 1999.

[Knapp 1999] Knapp, A.: "A Formal Semantics for UML Interactions", *UML 1999*, Lecture Notes in Computer Science, Vol. 1723, Springer-Verlag, October 1999.

[Kröger 1987] Kröger, F., *Temporal Logic of Programs*, Springer-Verlag, 1987.

[Manna and Pnuelli 1992] Manna, Z. and Pnuelli, A., *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, 1992.

[Moreira and Araújo 2000] Moreira, A., Araújo, J., "Generating Object-Z Specifications from Use Cases", *Enterprise Information Systems*, ed. J. Filipe, Kluwer Academic Publishers, 2000.

[Moreira and Clark 1996] Moreira, A., Clark, R., "Adding Rigour to Object-Oriented Analysis", *Software Engineering Journal*, Vol. 11(5), 1996.

[Moszkowski 1986] Moszkowski, B., *Executing Temporal Logic Programs*, Cambridge University Press, 1986.

[Overgaard 1999] Overgaard, G., "A Formal Approach to Collaborations in the Unified Modeling Language", *UML 1999*, Lecture Notes in Computer Science, Vol. 1723, Springer-Verlag, October 1999.

[OMG 2005] OMG: UML Resource Page, http://www.uml.org/, 2005.

[Warmer and Kleppe 2003] Warmer, J., Kleppe, A., *The Object Constraint Language: Getting Your Models Ready for MDA*, Second Edition, Addison-Wesley, 2003.

[Z.100 1994] Z.100, "Specification and Description Language SDL", *ITU-T*, June 1994.

[Ziemann and Gogolla 2003] Ziemann, P. and Gogolla, M., "OCL Extended with Temporal Logic", *PSI 2003*, Lecture Notes in Computer Science, Vol. 2890, Springer-Verlag, 2003.