Specifying Use Case Sequencing Constraints using Description Elements

Stéphane S. Somé
School of Information Technology and Engineering (SITE) University of Ottawa
800 King Edward, P.O. Box 450, Stn. A Ottawa, Ontario, K1N 6N5, Canada
ssome@site.uottawa.ca

Abstract

The ability to express sequencing constraints is critical to use case based software development. These constraints are needed to effectively compose use case behavior and support verification and validation activities such as simulation and testing.

In this paper, we consider the addition of use case description elements to explicitly express sequencing constraints between use cases. We introduce two complementary constructs. One allows to specify which use cases need to precede a use case and how these preceding use cases are synchronized. The second construct allows to specify which use cases are enabled from a use case and how these use cases execute concurrently. We relate the introduced elements to UML activity diagrams and implicit sequencing based on preconditions and postconditions.

1. Introduction

Use case modeling is considered an effective approach for requirements elicitation and analysis. Use cases allow functional requirements to be expressed from users point of view as a set of interactions in an intuitive way. The nature of use cases makes them well suited for user requirements capture. At the same time, with little structuring and formalization, use cases can also serve as functional requirements specification. Use case modelling thus constitutes a promising approach for bridging the gap between users and specifications.

The possibility for validation is an invaluable benefit for any requirements engineering approach. Validation aims at getting confidence that a specification effectively captures users concerns and needs. Typical validation approaches include simulation and acceptance testing. Use cases supports both approaches. Because they are description of users interactions, use cases are well suited to simulation. A use case based requirements specification can be validated by showing users that it captures their required interactions with the envisioned system rightfully [12]. Use cases are also good sources for acceptance test cases as illustrated by approaches such as [3, 10, 1].

An impediment to use cases based requirements validation is the necessity to combine sequentially related use cases in a global executable behavior model. For instance in a library system it is clear that a use case which goal is to perform a customer registration would have to be completed before a use case which goal is to hire books is possible. Similarly books need to be hired before they can be returned. This type of use case sequencing can not be directly expressed using UML use case relationships (*include*, *extend* and *generalization*). An attempt to force sequencing using these UML relationships results in poorly written use case models that suffer from the functional decomposition problem [2].

In different approaches [10, 14], graph-based models similar or derived from UML activity diagrams [9] are used to explicitly express use cases sequencing constraints. Other approaches including our previous work specify these sequencing constraints in a more implicit way using *preconditions* and *postconditions* [11, 8]. In this paper we consider an alternative approach for the expression of use case sequencing constraints. We exploit the potential of extension offered by use case templates to explicitly specify how use cases need to execute each in relation to the others. We compare our use case sequencing constructs to activity diagram and pre/postconditions based approaches.

The remainder of this paper is organized as follow. We introduce use case description as well as a running example in the next section. In section 3, we present our use case sequencing elements and compare with UML activity diagram as well as pre/postconditions based sequencing. We discuss some related works in section 4. Finally section 5 concludes the paper.

2. Use Case Description

Use cases related to a system are described in a use case diagram [9] that shows use case names, actors, relationships

Sixth International Workshop on Scenarios and State Machines (SCESM'07)
0-7695-2958-5/07 $20.00 © 2007 IEEE

between actors and use cases, and relationships between use cases. As an example, Figure 1 describes a UML use case model for an "Online Broker System". The goal of the

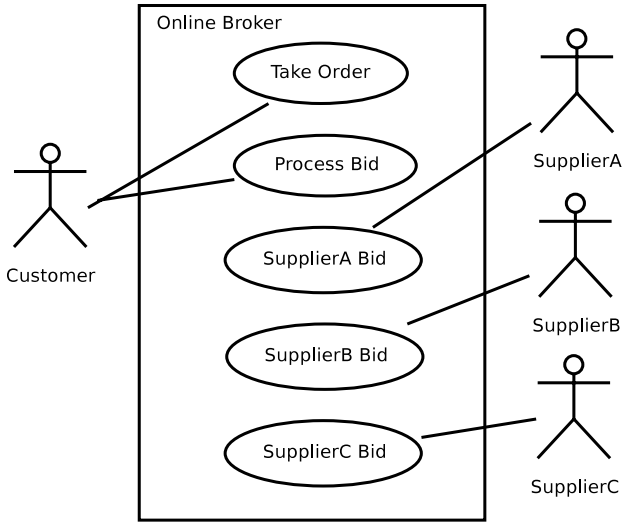


Figure 1. UML representation of a use case diagram for an Online Broker System.

system is to allow customers to find the best supplier for a given order. A customer fills up an online order form and after submission, the system broadcast it to suppliers. We assume three suppliers in this example, "SupplierA", "SupplierB" and "SupplierC". Each supplier after examining the order may decide to decline or submit a bid. Submitted bids are sent back to the broker to be shown to the customer, who eventually ask the system to proceed with a bid.

Different templates have been proposed for use cases description. These templates propose different sections for recording use case related information. Examples of sections are: *title*, *system under design*, *precondition*, *postcondition* and *steps*. Figure 2 shows a description of the use cases in the "Online Broker System" using our use case template introduced in [11].

Two main parts can be distinguished in a use case description. A *static part* consisting of sections title, system under design, precondition, postcondition and a *dynamic part* consisting of sections Steps and Alternatives. The *dynamic part* describes interactions between the system under consideration and actors in the environment. Beside events such as triggers from actors and system reactions, a use case dynamic part may also include directives such as the `use case inclusion` directive.

Each use case consists of a *primary scenario* and zero or more *secondary scenarios*. The primary scenario is the sequence of events in the section titled "*Steps*". Secondary

Title: Take Order
System Under Design: Broker System
Precondition: System is online
Steps
1.Customer loads the order page,
creates an order by filling a list of items
2.Customer submits the order
3.System broadcast the order
Success Postcondition: The Order has been broadcasted

Title: SupplierA Bid
System Under Design: Broker System
Precondition: An Order has been broadcasted
Steps
1.SupplierA receives the order and examines it
2.SupplierA submits a bid for order
3.The System shows the Bid to the Customer
Alternatives
1.a.SupplierA can not satisfy the Order
1.a.1.SupplierA passes on the Order
Success Postcondition: SupplierA has submitted a bid

Title: SupplierB Bid
System Under Design: Broker System
Precondition: An Order has been broadcasted
Steps
1.SupplierB receives the order and examines it
2.SupplierB submits a bid for order
3.The System shows the Bid to the Customer
Alternatives 1.a.SupplierB can not satisfy the Order
1.a.1.SupplierB passes on the Order
Success Postcondition: SupplierB has submitted a bid

Title: SupplierC Bid
System Under Design: Broker System
Precondition: An Order has been broadcasted
Steps
1.SupplierC receives the order and examines it
2.SupplierC submits a bid for order
3.The System shows the Bid to the Customer
Alternatives
1.a.SupplierC can not satisfy the Order
1.a.1.SupplierC passes on the Order
Success Postcondition: SupplierC has submitted a bid

Title: Process Bids
System Under Design: Broker System
Precondition: SupplierA has bidded or SupplierB has bidded or SupplierC has bidded
Steps
1.Customer examines the bid
2.Customer signals the system to proceed with bid
3.System put an order with the selected bidder

Figure 2. Use cases for the Online Broker System.

0-7695-2958-5/07 $20.00

scenarios include events from the section titled "*Steps*" followed by events in the section titled "*Alternatives*". For instance, use case "SupplierA Bid" includes a primary scenario and one secondary scenario. The primary scenario consists of the sequence of steps: *1-2-3*, while the secondary scenario consists of: *1-1.a.1*.

A use case *execution* corresponds to an instantiation of one of its scenarios. For instance, use case "SupplierA Bid" execution would result in the instantiation of one of the two scenario above with the occurrence of the corresponding sequence of events.

A use case model does not show use cases sequencing constraints. For instance, the use cases in the Broker example are sequentially related. Use cases "SupplierA Bid", "SupplierB Bid" and "SupplierC Bid" are supposed to execute concurrently following use case "Take Order". An instance of use case "Process Order" may be started following any of use cases "SupplierA Bid", "SupplierB Bid" or "SupplierC Bid". However these relations can not be modeled in the use case diagram shown in Figure 1 because there is no provision in the UML for the specification of such sequencing relations. Preconditions and postconditions can be used to express some of these relations. However, by only relying on preconditions and postconditions, some nuances such as the distinction between concurrent and alternative execution can not be expressed. Moreover, according to our experience, use case modelers have difficulty finding pre and postconditions at the early stages of use case modeling. Preconditions and postconditions are often too broad or too restrictive. On the other hand there is usually a clear idea of how use cases should follow each other.

3. Use case sequencing elements

We propose to enhance use case description with elements allowing to express sequencing relations. We first present our use case sequencing requirements. Then, we introduce two use case sequencing constructs and compare to UML activity diagram as well as pre/postconditions based sequencing.

3.1. Use case sequencing requirements

The following are our initial requirements for use case sequencing.

Req1 It should be possible to specify which use cases must precede a given use case.

Req2 It should be possible to specify how the use cases preceding a use case are synchronized in relation to that use case.

Req3 It should be possible to specify which use case is enabled to execute by the execution of a use case.

Req4 When more than one use cases are enabled by a use case, it should be possible to specify whether these use cases execute concurrently or alternatively.

These requirements should be considered as minimal requirements for modelling the most common use case sequencing situations such as the ones in the "Online Broker System" example. They cover situations related to *sequential* execution, *iterative* execution, *concurrent* execution and *alternative* execution. We do not consider data flow issues.

3.2. Use case sequencing constructs

We introduce two constructs for expressing sequencing constraints in use case descriptions. Figure 3 shows the use cases in the Online Broker example rewritten to include these sequencing constructs[1].

For requirement Req1, we introduce a section in use cases *static part* titled "Follows" (*follow* section). The *follow* section introduces a set of use cases that need to be executed before the described use case. For instance, use case "SupplierA Bid" *follows* section refers to use case "Take Order" to express that an execution of "Take Order" must precede an execution of "SupplierA Bid".

In order to satisfy requirement Req2, the use cases in a *follow* section are specified as an expression reflecting how they should be synchronized in relation to the described use case. We use the two boolean operators AND and OR with the following meaning. Operator AND expresses synchronization while OR captures asynchronism. Given use cases $uc_0, uc_1, uc_2, \cdots, uc_n$, the following interpretation is given.

- If the follows section of uc_0 is specified as "$uc_1\, AND\, uc_2\, AND\, \cdots uc_n$", all of uc_1, uc_2, $\cdots$, uc_n must reach a point from which use case uc_0 is *enabled* before use case uc_0 (synchronism).

- If the follow section of uc_0 is specified as "$uc_1\, OR\, uc_2\, OR\, \cdots uc_n$", use case uc_0 may be executed as soon as any of use cases uc_1, $\cdots$, uc_n reach a point from which use case uc_0 is *enabled* (asynchronism).

  For instance use case "Process bids" follows section "SupplierA Bid OR SupplierB Bid OR SupplierC Bid" interpretation is that "Process bids" can be executed whenever any of use cases "SupplierA Bid", "SupplierB Bid" or "SupplierC Bid" has reached a point where "Process Bids" is enabled. In other words, the three use cases execution is not synchronized.

Operators AND and OR may be combined to express more complex sequential relations.

[1] We only show use case "SupplierA Bid" as "SupplierB Bid" and "SupplierC Bid" are similar.

0-7695-2958-5/07 $20.00

Title: Take Order
System Under Design: Broker System
Precondition: System is online
Steps
1.Customer loads the order page,
creates an order by filling a list of items
2.Customer submits the order
3.System broadcast the order
4.`Enable in parallel`: SupplierA Bid, SupplierB Bid, SupplierC Bid
Success Postcondition: The Order has been broadcasted

Title: SupplierA Bid
System Under Design: Broker System
`Follows: Take Order`
Precondition: An Order has been broadcasted
Steps
1.SupplierA receives the order and examines it
2.SupplierA submits a bid for order
3.The System shows the Bid to the Customer
4.`Enable`: Process Bids
Alternatives
1.a.SupplierA can not satisfy the Order
1.a.1.SupplierA passes on the Order
Success Postcondition: SupplierA has submitted a bid

Title: Process Bids
System Under Design: Broker System
`Follows: SupplierA Bid` OR `SupplierB Bid` OR `SupplierC Bid`
Precondition: SupplierA has bidded or SupplierB has bidded or SupplierC has bidded
Steps
1.Customer examines the bid
2.Customer signals the system to proceed with bid
3.System put an order with the selected bidder

Figure 3. Use cases for the Online Broker System enhanced with use case sequencing elements.

For requirement Req3, we introduce a directive similar to the `use case inclusion` directive. We refer to this directive as a `use case enabling` directive as it allows to explicitly state which use case is enabled for execution. For instance, use case "Process Bids" is enabled in step *4* of use case "SupplierA Bid". An execution of "Process Bids" becomes therefore possible from that point according to this use case *follows* section.

We distinguish a parallel variant of the `use case enabling` directive in order to satisfy requirement Req4. This variant allows stating that enabled use cases execute all concurrently. For instance step *4* of use case "Take Order" specifies that "SupplierA Bid", "SupplierB Bid" and "SupplierC Bid" are enabled in *parallel*. As a result, a scenario from each of the use cases might proceed concurrently with the others. The non-parallel variant of the `use case enabling` directive may also specify more than one use case. In that case however, the other use cases are disabled as soon as one of the use case starts executing. In order to avoid non deterministic situations, all use cases involved in a non-parallel `use case enabling` directive must have a distinct triggering event. More formally, firstEvent and trigger being two functions such that given a use case *uc* firstEvent(uc) is the first event of *uc* and trigger(*ev*) returns true if *ev* is a trigger event, the following rule needs to be satisfied.

Rule 1 *Given* UC = $\{uc_1, \cdots, uc_n\}$ *a set of use cases referred to in a non-parallel* `enabling` *directive,* $\forall uc_i \in UC$, *trigger*(*firstEvent*(uc_i)) *is* true *and* $\nexists\ uc_j\ (i \neq j) \in$ UC *such that* *firstEvent*(uc_i) = *firstEvent*(uc_j).

A `use case enabling` directive may appear at any point in a use case. In a situation where an enabling directive is not the last element of a scenario, all events following that directive are concurrent with the enabled use cases. We also allow a use case to enable itself. Thus modelling an iterative behavior.

Follow sections and use case enabling directives depend each on the other. The following rule must be satisfied for consistency.

Rule 2 *When a use case* uc_i *refers to a use case* uc_j *in its* follow *section, use case* uc_j *must include an enabling directive referring to use case* uc_i. *Conversely, for each use case referred to in an enabling directive, the enabled use case must include the enabling use case in its* follow *section.*

Both constructs are necessary to fully satisfy requirements Req1 to Req4. The use of a *Follow* section alone would not allow to express concurrency (Req4) additionally, it wouldn't be possible to specify at which point a use case becomes possible. Conversely, the use case enabling directive alone would not make it possible to specify how the use cases preceding a use case should be synchronized (Req2).

3.3. Comparison with UML activity diagrams

Figure 4 shows how our use case sequencing constructs are related to UML activity diagrams [9], and Figure 5 shows a UML activity diagram representation of the sequencing relations specified in Figure 3. Figure 4 only considers enabling directives at the end of scenarios. We can not directly map situations where enabling directives are not at the end of scenarios to activity diagrams because

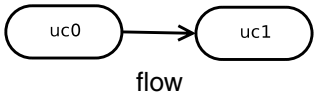
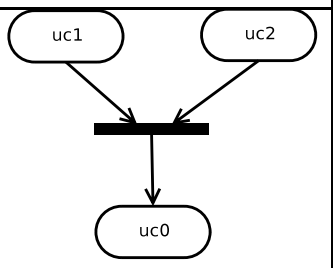
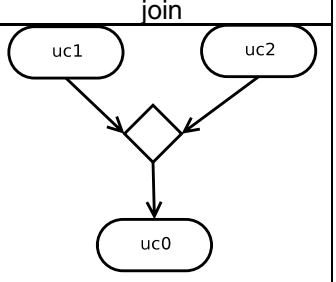
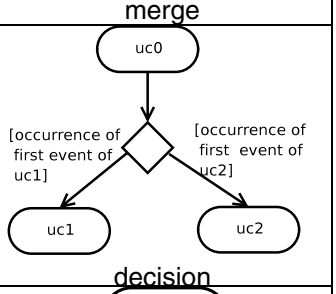
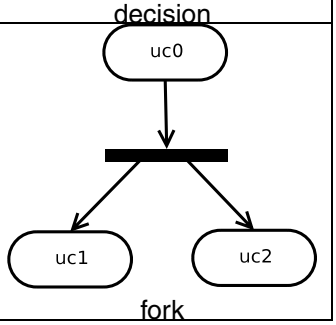
Use Case Construct	Corresponding Activity Diagram
Title: uc0 1. ... ... n. Enable: uc1 Title: uc1 ... Follows: uc0 (a)	
Title: uc0 ... Follows: uc1 AND uc2 (b)	
Title: uc0 ... Follows: uc1 OR uc2 (c)	
Title: uc0 1. ... ... n. Enable: uc1, uc2 (d)	
Title: uc0 1. ... ... n. Enable in parallel: uc1, uc2 (e)	

Figure 4. Mappings between use case sequencing constructs and UML activity diagrams.

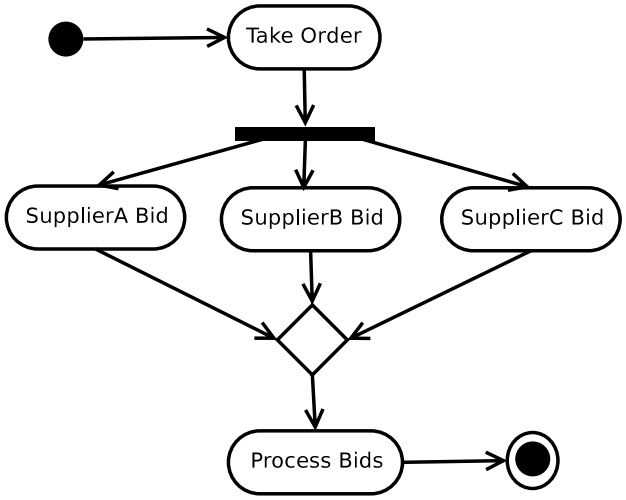


Figure 5. Description of Figure 3 sequencing relations as a UML activity diagram.

of the level of granularity of actions in activity diagrams. In order to describe situations where enabling directives are not the last elements of scenarios using UML activity diagrams, the use cases need to be rewritten such that all use case enabling directives are last in their scenario.

An enabling directive that refers to a single use case (with its matching *follow* section) corresponds to a simple flow from one use case to the other (Figure 4-a). Situations where a *follow* section refers to more than one use case correspond to a join when operator AND is used (Figure 4-b), or a merge when operator OR is used (Figure 4-c). We can infer from Figure 4-b and 4-c that a situation involving a combination of operators AND and OR corresponds to a sequencing of join and merge nodes. Figures 4-d and 4-e show that a non-parallel use case enabling directive corresponds to a decision modelling a deferred choice [13] based on the next event occurrence, while a parallel use case enabling directive corresponds to a fork. Decisions involving conditions can be modeled by embedding enabling directives in different alternatives of a use case.

3.4. Comparison with preconditions and postconditions

Most use case templates include sections for preconditions and postconditions. A use case precondition is a *condition* that needs to hold before the use case, while a use case postcondition is a *condition* that is guaranteed to hold at the end of the use case. We distinguish *success* postconditions that hold at the end of primary scenarios from *alternative* postconditions that hold at the end of secondary scenarios. Preconditions are assumed to be verified before

0-7695-2958-5/07 $20.00

a use case is invoked. They usually result from the execution of other use cases. As such, preconditions and postconditions are implicit specification of use case sequencing constraints.

A connection can be made between our use case sequencing constructs and pre/postconditions. Assuming functions pre and post such that: pre(uc) is the precondition of a use case *uc* and given a scenario *sc* in use case *uc*, post(sc) is the postcondition associated with scenario *sc*. The inclusion of use case uc_i in the *follow* section of use case uc_j (and/or the presence of a use case enabling directive referring to use case uc_j as the last element of scenario sc_i) correspond to having $\mathsf{post}(sc_i) \Rightarrow \mathsf{pre}(uc_j)$ true, with sc_i a scenario in use case uc_i.

Preconditions and postconditions allows to specify which use case must precede a given use case (Req1) and which use cases are enabled after a use case (Req3). However, they do not offer a possibility to specify how use cases are synchronized (Req2) or whether several enabled use cases execute concurrently or alternatively (Req4). For instance, by analyzing the pre and postconditions of use cases in Figure 2 , we can infer that use case "Take Order" could be followed by use cases "SupplierA Bid", "SupplierB Bid" and "SupplierC Bid". However, there is no indication to how these use cases would execute. Similarly we can infer the sequencing relations between use cases "SupplierA Bid", "SupplierB Bid", "SupplierC Bid", and "Process Bids" without any information on how they are synchronized. Additionally, since postconditions are only checked at the end of a scenario, situations where enabling directives are not the last elements of scenarios do not have an equivalent.

We do not suggest a replacement of implicit sequencing provided by preconditions and postconditions with our use case sequencing constructs. Beside allowing to express some sequencing constraints, preconditions and postconditions play an important role in use case documentation. They provide a description of system *states* before and after the execution of use cases as *conditions*. Use case sequencing is only implicitly inferred from matching these states. The relation of *states* and *conditions*, with an extension of pre/postconditions to the description of operations are the basis for state machine synthesis from use cases [11]. In that context, the full extent of use case sequencing description capabilities obtained by our sequencing constructs can be implicitly obtained from pre/postconditions[2]. However, there is still a drawback with preconditions and postconditions as use case sequencing description mechanisms. Information needed to accurately specify pre/postconditions is usually not evident at the early stages of use case modelling. This makes pre/postconditions specification an error prone process.

[2]Further discussion on this topic is out of the scope of this paper.

We suggest using implicit sequencing constructs and pre/postconditions in conjunction. Typically, the requirements and assumptions about use cases sequencing would be transcribed as early as possible using the explicit sequencing constructs. These requirements and assumptions would then be elaborated using pre and postconditions when more information become available. Explicit constraints specified by use case sequencing constructs and implicit constraints derived from pre/postconditions must comply with the following consistency rules. We assume functions pre and post as above.

Rule 3 *Given use cases uc_i and uc_j, if uc_i is included in the* follow *section of uc_j, there must exist a scenario sc_i in uc_i such that* $\mathsf{post}(sc_i) \Rightarrow \mathsf{pre}(uc_j)$ *(notice that in accordance with rule 2, scenario sc_i must include an enabling directive referring to use case uc_j).*

Rule 4 *Given use cases uc_i and uc_j, if there is a scenario sc_i in uc_i such that* $\mathsf{post}(sc_i) \Rightarrow \mathsf{pre}(uc_j)$*, use case uc_i must be included in the* follow *section of uc_j and there must be a use case enabling directive referring to uc_j in scenario sc_i.*

4. Related work

Different approaches related to the specification of scenario and use case sequencing constraints are based on graphical models where nodes are scenarios/use cases and edges capture information on how the scenarios or use cases execute in sequence. The ITU-T recommendation Z.120 [6] introduced High-Level MSC (HMSC) to graphically model Message Sequence Charts sequencing. Interaction Overview Diagrams [9] in the UML offer a mechanism similar to HMSC for the expression of sequencing relations between UML Sequence Diagrams.

Glinz [4] proposed an approach for use case composition where use case sequencing relations (sequence, alternative, iteration and concurrency) are captured in a derivative of Jackson diagram. The TOTEM approach [1] uses a UML activity diagram to model use case sequencing for system level test cases derivation. In [14], a graphical notation called "Use Case Charts" is proposed for use case specification. Use case charts represent use case models over three levels. Levels *1* and *2* are variants of UML activity diagrams used to model sequencing relations at the use cases level and at the scenarios level respectively. The SCENT approach [10] also makes use of a graphical model. The model referred to as a "Dependency Chart" allows to graphically specify use cases sequencing constraints. Dependency Charts are also used to describe other dependencies such as data and resource dependencies.

Our main criticism of graphical models including UML activity diagrams boils down to the granularity of ele-

0-7695-2958-5/07 $20.00

ments. Graphical models specify relations between whole use cases. However, sequencing is usually dependent on scenarios within use cases. For instance, in an online ordering system, a *primary* scenario in a user identification use case would likely enable use cases allowing products ordering, while *secondary* scenarios that describe unsuccessful identification would enable other use cases. Other limitations as discussed in Section 3.3, include the inability to specify precisely when a use case becomes enabled.

An advantage offered by graphical approaches is that they provide a visual depiction of use case sequencing relations. This allows to quickly validate sequencing assumptions by inspection. Mappings between use case sequencing constructs and UML activity diagrams presented in Figure 4 and illustrated in Figure 5 allow to envision automated derivation of UML activity diagram depicting sequencing relations from use cases augmented with sequencing constructs.

We extended a use case description template to include the possibility to specify sequencing constraints. Different templates include sections similar to our *follow* section [5]. The basic objective of the *follow* section is to provide an ability to specify which use cases precede (or follow) a given use case. However, to the best of our knowledge, no template formally defines the content of that section to the degree of this paper. We are also not aware of any previous work on use case enabling directive.

5. Conclusions

In this paper we presented two complementary constructs for the specification of use case sequencing constraints. These constructs allow to express common use case dependencies such as sequential execution, alternative execution, concurrent execution and iterative execution without relying on an additional model. We defined mappings between our use case sequencing constructs and UML activity diagrams that would allow synthesis of the later from the former and consistency checking.

Use case sequencing constraints capture is an important step toward use case composition and global behavior synthesis. In our previous work, we developed a toolset for use case based requirements definition and elaboration called Use Case Editor (UCEd) [12]. We updated UCEd template and language with the use case sequencing constructs discussed in this paper. We also implemented use case verification routines for the consistency rules presented in section 3. Given a use case model, these verification routines allows to check that none of the rules is violated and provide guidance for the removal of eventual inconsistencies. We are also developing a state machine synthesis approach based on our sequencing constructs. Our goal is to allow early validation of use case based requirements by providing facilities for simulation. The obtained state model would also be useful for the validation of design and implementation artefacts by the mean of testing.

The types of sequencing constraints supported in this paper are not exhaustive. Other useful types of constraints such as *preemption* [7] exist. In our future work, we will consider ways to extend use case description to support additional sequencing constraints.

References

[1] L. Briand and Y. Labiche. A UML-Based Approach to System Testing. Technical Report TR SCE-01-01, Carleton University, 2002.

[2] M. Fowler. www.DistributedComputing.com, 1998.

[3] P. Fröhlich and J. Link. Automated test case generation from dynamic models. In E. Bertino, editor, *ECOOP 2000 - Object-Oriented Programming, 14th European Conference*, pages 472–492. Springer, 2000.

[4] M. Glinz. An Integrated Formal Model of Scenarios Based on Statecharts. In *Software Engineering - ESEC'95. Proceedings of the 5th European Software Engineering Conference*, pages 254–271. Springler LNCS 989, 1995.

[5] A. Holub. OO design process: Use cases, an introduction. http://www-128.ibm.com/developerworks/library/co-design5.html, 2000.

[6] ITU-T. Recommendation Z120, Message Sequence Charts, 2004.

[7] I. Krüger. Modeling and Synthesis with MSC Extensions for Broadcasting, Overlapping, Preemptive, and Triggered Collaborations. In *Scenarios and state machines: models, algorithms, and tools (SCESM'03)*, 2003.

[8] C. Nebut, F. Fleurey, Y. L. Traon, and J.-M. Jézéquel. Automatic Test Generation: A Use Case Driven Approach. *IEEE Transactions On Software Engineering*, 32(3):140–155, march 2006.

[9] OMG. UML 2.0 Superstructure, 2003. Object Management Group.

[10] J. Ryser and M. Glinz. Using dependency charts to improve scenario-based testing. In *Proceedings of 17th International Conference on Testing Computer Software (TCS2000)*, 2000.

[11] S. Somé. Beyond Scenarios: Generating State Models from Use Cases. In *Scenarios and state machines: models, algorithms, and tools (SCESM'02)*, 2002.

[12] S. Somé. Supporting Use Cases based Requirements Engineering. *Information and Software Technology*, 48(1):43–58, 2006.

[13] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[14] J. Whittle. Specifying Precise Use Cases with Use Case Charts. In *Satellite Events at the MODELS 2005 Conference*, volume 3844 of *LNCS*, pages 290–301. Springer-Verlag, 2005.