Specifying Precise Use Cases with Use Case Charts

Jon Whittle

Dept of Information & Software Engineering
George Mason University
4400 University Drive
Fairfax, VA 22030
jwhittle@ise.gmu.edu

Abstract. Use cases are a popular method for capturing and structuring software requirements. The informality of use cases is both a blessing and a curse. It enables easy application and learning but is a barrier to automated methods for test case generation, validation or simulation. This paper presents *use case charts*, a precise way of specifying use cases that aims to retain the benefits of easy understanding but also supports automated analysis. The graphical and abstract syntax of use case charts are given, along with a sketch of their formal semantics.

1 Use Case Charts

Use cases are a popular way of structuring and analyzing software requirements but are usually written informally as a set of use case diagrams and text-based templates. This makes them very easy to use but is a barrier to the application of automated analysis methods such as test case generation, simulation and validation. More precise formalisms for specifying use cases are needed but the advantages of informal notations should not be sacrificed in the process. In this paper, *use case charts*, a 3-level notation based on extended activity diagrams, is proposed as a way of specifying use cases in detail, in a way that combines the formality of precise modeling with the ease of use of existing notations. The primary purpose of use case charts so far has been to simulate use cases but use case charts are also precise enough for test case generation and automated validation. With respect to simulation, use case charts provide sufficient detail that a set of communicating finite state machines can be generated automatically from them. These state machines can then be simulated using existing state machine simulators.

The idea behind use case charts is illustrated in Figure 1. For the purposes of this paper, a use case is defined to be a collection of scenarios, where a scenario is an expected or actual execution trace of a system. A use case chart specifies the scenarios for a system as a 3-level, use case-based description: level-1 is an extended activity diagram where the nodes are use cases; level-2 is a set of extended activity diagrams where the nodes are scenarios; level-3 is a set of

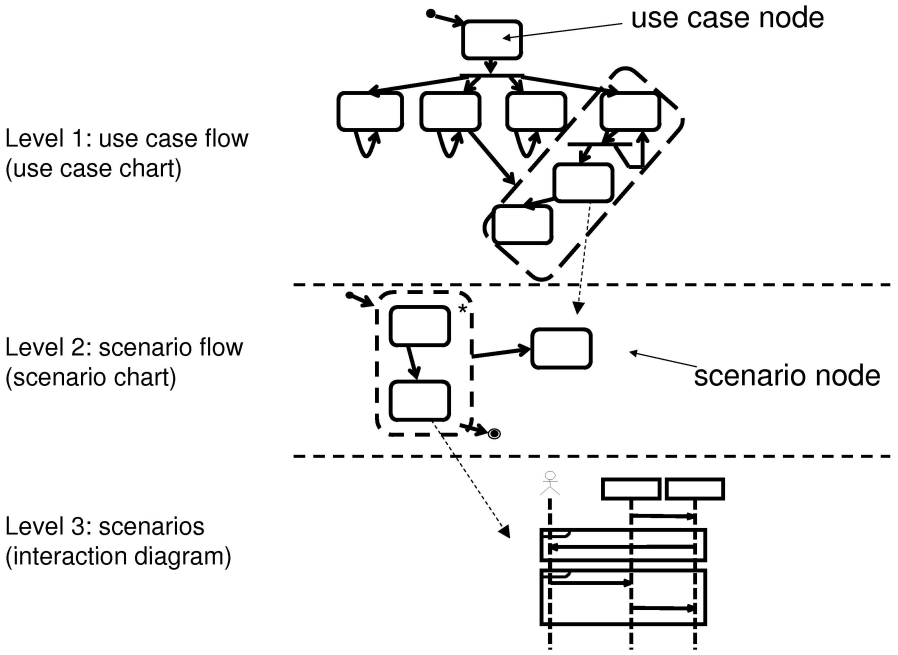

Fig. 1. Use Case Charts.

UML2.0 ([UML05]) interaction diagrams. Each level-1 use case node is defined by a level-2 activity diagram (i.e., a set of connected scenario nodes). This diagram is called a *scenario chart*. Each level-2 scenario node is defined by a UML2.0 interaction diagram.

Semantically, control flow of the entire use case chart starts with the initial node of the use case chart (level 1). When flow passes into a use case chart node at level-1, the defining level-2 scenario chart is executed with flow passing from the scenario chart's initial node in the usual manner. Flow exits a scenario node when a final node is reached. Note that there are two types of final nodes for scenario charts: those that represent successful completion of the scenario chart and those that represent completion with failure. Flow only continues beyond a scenario chart if a final success node is reached. If a final failure node is reached, the use case thread to which the scenario chart belongs is terminated. A formal semantics for use case charts is sketched in Section 3.

Figures 2, 3 and 4 give an example of how use case charts can be used to precisely describe use cases. The system under development is an automated train shuttle service in which autonomous shuttles transport passengers between stations [SEG]. When a passenger requires transport, a central broker asks all

active shuttles for bids on the transport order. The shuttle with the lowest bid wins. A complete set of requirements for this application is given in [SEG]. Figure 2 shows a use case chart that includes use cases for initialization of the system, maintenance and repair of shuttles, and transportation (split into multiple use cases). Figure 3 is a scenario chart that defines the Carry Out Order use case. Figure 4 is an interaction diagram forming part of the definition of the use case Make A Bid.

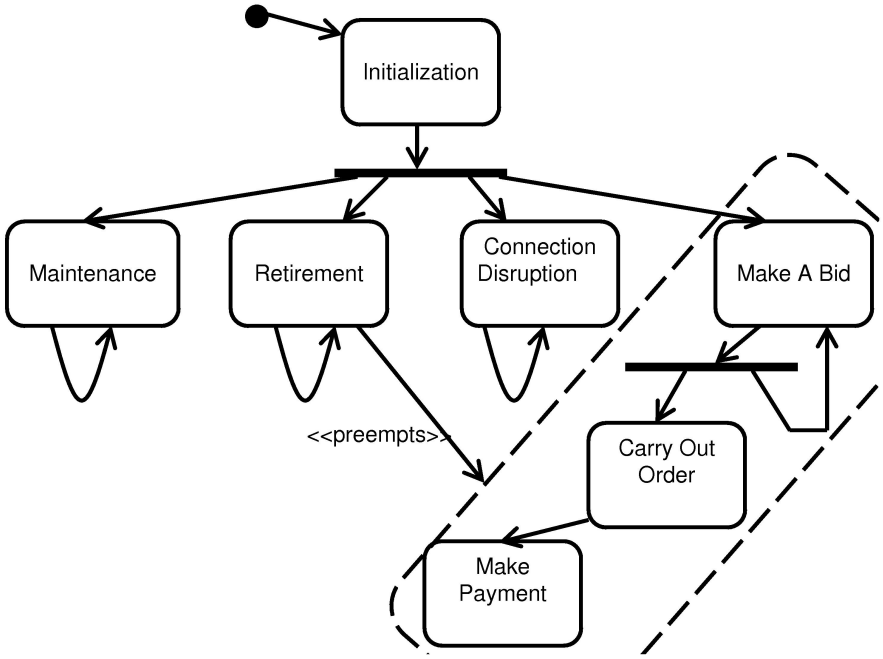

Fig. 2. Shuttle System Use Case Chart.

The use case chart in Figure 2 shows the main use cases for the shuttle system and the relationships between them. As stated previously, a use case chart is an extended activity diagram. Note that the usual ⟨⟨includes⟩⟩ and ⟨⟨extends⟩⟩ relationships from use case diagrams are not part of use case charts. If desired, these can be represented independently on a conventional use case diagram. Figure 2 shows that the shuttle system first goes through an Initialization use case. After that, four use cases execute in parallel. If the Make A Bid use case is successful, it can be followed by Carry Out Order or another bidding process (executed in parallel). The Retirement use case represents the case when the shuttles are shut down. It preempts any activity associated to Make A Bid. This

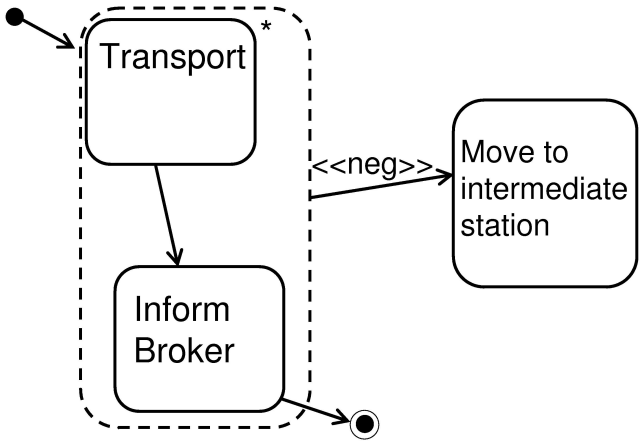

Fig. 3. Shuttle System Scenario Chart for Carry Out Order.

is represented by a stereotyped preemption relationship that applies to a region. A region is a set of nodes enclosed in a dashed box. Note that, in the figure, a region is syntactic sugar and can be replaced by multiple preemption arrows, one to each node in the region.

Figure 3 is a description of what happens in the Carry Out Order use case. Transportation of passengers takes place and the broker is informed of success. The asterisk in the region represents the fact that the region may execute in parallel with itself any numbers of times. In other words, the use case may involve multiple concurrent transports of passengers. However, the requirements of the problem state that during transport, shuttles may not move to intermediate stations except to pick up or drop off passengers. This is captured by introducing a negative scenario node with a stereotyped negation arrow. Note that scenario charts must have at least one final success or final failure node. A final success node represents the fact that execution of the use case has successfully completed and is given graphically by the final activity node as in Figure 3. A final failure node says that the use case completes but that execution should not continue beyond the use case. This is given graphically using the final flow node of activity diagram notation, i.e., a circle with a cross through it[1]. As an example, suppose that the passenger transport cannot be completed for some reason. This could be captured by introducing a scenario capturing the failure and then an arrow to a final failure node. In this case, when the final failure node is reached, the

[1] Note that this is not the standard UML2.0 interpretation for the final flow node.

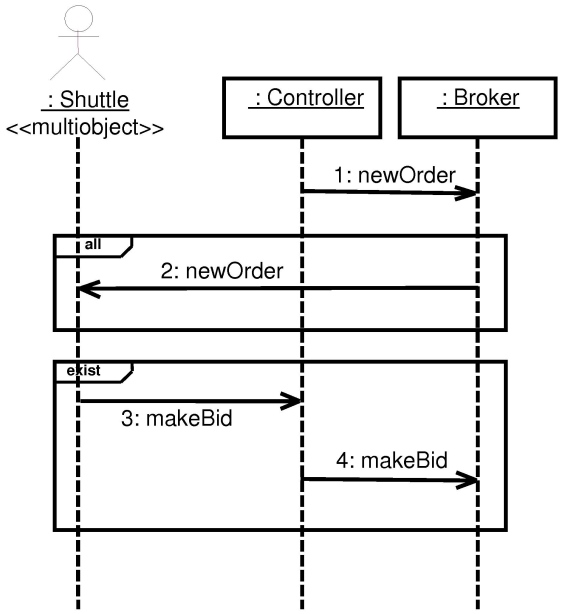

Fig. 4. Shuttle System Interaction Diagram for a scenario in Make A Bid.

Make Payment use case in Figure 2 will not execute — i.e., payment will not be paid for an unsuccessful transport.

Each scenario node in Figure 3 is described by a UML2.0 interaction diagram. Figure 4 shows an interaction diagram that is part of the Make A Bid use case. This particular example is shown to illustrate extensions that use case charts introduce to UML2.0 interaction diagrams, namely, multiobjects and universal/existential messages. We introduce two new interaction operators, exist and all. We also introduce a stereotype $\langle\langle$multiobject$\rangle\rangle$ which denotes that an interaction applies to multiple instances of a classifier. In the figure, Shuttle is stereotyped as a multiobject which means that multiple shuttles may participate in the interaction. There are two interaction fragments. The first has operator "all". This means that the Broker sends the enclosed messages to all shuttles. The second operator has operator "exist" meaning that there must be at least one makeBid message to Controller followed by at least one makeBid message to Broker. The semantics can be easily extended to more than one multiobject. For example, if in the "all" fragment, Broker is also a multiobject, then legal traces would be those in which each broker sends a message to each shuttle.

The activity diagrams used in use case charts and scenario charts are a restricted version of UML2.0 activity diagrams but with some additional relationships between nodes. They are restricted in the sense that they do not include object flow, swimlanes, signals etc. They do include additional notations, however. The abstract syntax is defined in Section 2. The concrete syntax reuses as much of the activity diagram notation as possible. Informally, the allowed arrow types between nodes (either in use case or scenario charts) are given as follows, where, for each arrow, X and Y are either both scenario nodes or both use case nodes:

1. X continues from Y (i.e., the usual activity diagram arrow)
2. X and Y are alternatives (the usual alternative defined by a condition in an activity diagram)
3. X and Y run in parallel (the usual activity diagram fork and join)
4. X preempts Y — i.e., X interrupts Y and control does not return to Y once X is complete. This is shown graphically by an arrow stereotyped with $\langle\langle$preempts$\rangle\rangle$ from X to Y.
5. X suspends Y — i.e., X interrupts Y and control returns to Y once X is complete. This is shown graphically by an arrow stereotyped with $\langle\langle$suspends$\rangle\rangle$ from X to Y.
6. X is negative — i.e., the scenarios defined by X should never happen. This is shown graphically by an arrow stereotyped with $\langle\langle$neg$\rangle\rangle$ to X and where the source of the arrow is the region over which the scope of the negation applies.
7. X may have multiple copies — i.e., X can run in parallel with itself any number of times. This is shown graphically by an asterisk attached to node X.
8. X crosscuts Y — X is an aspect that crosscuts Y. This is shown graphically by an arrow stereotyped with $\langle\langle$aspect$\rangle\rangle$ from X to Y.

Discussion of aspects is outside the scope of this paper. The interested reader is referred to [WA04]. Briefly, a use case node is an aspect if it crosscuts other use cases. Similarly, a scenario node is an aspect if it crosscuts other scenarios. Use case charts contain well-defined notations for representing and composing aspects.

In addition, use case charts and scenario charts may have regions (graphically shown by dashed boxes) that scope nodes together. Arrows of type (4), (5), (8), in the preceding list, may have a region as the target of the arrow. Arrows of type (7) may have a region as the source of the arrow. All other arrows do not link regions.

Arrow types (4), (5), (6) and (8) are not part of UML2.0 activity diagrams (although there is a similar notation to (4) and (5) for interruption). Activity diagrams do have a notion of region for defining an interruptible set of nodes. Regions in use case charts are a general-purpose scoping mechanism not restricted to defining interrupts. Note that the semantics for use case charts is, in places, different than UML2.0 activity diagrams and is sketched in section 3.

In addition to the arrow and region extensions, there are minor extensions to interaction diagrams.

The graphical notation for use case charts is similar to notations such as UML2.0 interaction overview diagrams (IODs) and high-level message sequence charts (hMSCs). Use case charts are a hierarchical approach to defining use cases. In IODs, there are only two levels of hierarchy — activity diagrams connect references to interaction diagrams but use cases are not incorporated. In hMSCs, nodes can be references to other hMSCs so there is an unlimited number of hierarchical levels. However, all references in hMSCs ultimately are to interaction diagrams (MSCs) so, once again, the third use case level is not captured. Modeling use cases with either IODs or hMSCs would require, in addition, the usual use case diagrams. The use case chart approach also extends both activity diagrams and interaction diagrams to increase the expressive power of use case charts. The key goal of use case charts is to support use case simulation. This cannot be done with existing use case modeling notations.

2 Use Case Chart Syntax

2.1 Abstract Syntax for Scenario Charts

The abstract syntax of a scenario chart is given first.

Definition 1. *A* scenario chart *is a graph of the form* $(S, R_S, E_S, s_0, S_F, S_{F'}, L_S, f_S, m_S, L_E)$ *where* S *is a set of scenario nodes,* $R_S \subseteq \mathcal{P}(S)$ *is a set of regions,* $E_S \subseteq (\mathcal{P}(S \cup R_S) \times \mathcal{P}(S \cup R_S) \times L_E)$ *is a set of edges with labels from* L_E*,* $s_0 \in S$ *is the unique initial node,* $S_F \subset S$ *is a set of success final nodes,* $S_{F'} \subset S$ *is a set of failure final nodes,* L_S *is a set of scenario labels,* $f_S : S \to L_S$ *is a total, injective function mapping each scenario node to a label and* $m_s : S \cup R_S \to \{+, -\}$ *is a total function marking whether each scenario or region can have multiple concurrent executions. The labels in* L_S *are references to an interaction diagram.* L_E *is defined to be the set* $\{normal, neg, preempts, suspends\}$*.* L_S *is the set of words from some alphabet* Σ*.*

This definition describes a graph where edges may have multiple sources and targets. This subsumes the notion of fork and join from activity diagrams which can be taken care of by allowing edges to have multiple source nodes and/or multiple target nodes. Multiple source nodes lead in the use case chart graphical notation to a join and multiple target nodes lead to a fork.

Regions are a scoping mechanism used to group nodes. For the most part, they are simply syntactic sugar and can be eliminated by replacing each outgoing edge with outgoing edges for each node in the region, and each incoming edge with incoming edges for each node in the region. For example, a region with an incoming preemption edge can be replaced with edges that preempt each node in the region. A region with a normal outgoing edge (i.e., no stereotypes) is equivalent to normal edges leaving each node in the region, i.e., a join.

The only case when regions cannot be viewed as syntactic sugar is when a region is marked to have concurrent executions (graphically, an asterisk). This case cannot be eliminated (without changing the semantics) by projecting the concurrent executions inside the region.

As stated previously, the intuition behind success final and failure final nodes is that a success final node denotes successful completion of the scenario chart — and hence that the current "thread" in the enclosing use case chart continues; a failure final node denotes that the scenario chart completes but unsuccessfully — and hence that the current "thread" in the enclosing use case chart terminates.

This paper omits the notion of condition but it is enough to say that guards could be placed on arrows leaving a node.

2.2 Abstract Syntax for Use Case Charts

The abstract syntax for a use case chart is almost identical except that a use case chart has only one type of final node (for success) and each use case node maps to a scenario chart not an interaction diagram. Only one type of final node is required for use case charts because there is no notion of success or failure — either a use case chart completes or it does not.

Definition 2. *A* use case chart *is a graph of the form* $(U, R_U, E_U, u_0, U_F, C, f_U, m_U, L_E)$ *where* U *is a set of nodes,* $R_U \subseteq \mathcal{P}(U)$ *is a set of regions,* $E_U \subseteq (\mathcal{P}(U \cup R_U) \times \mathcal{P}(U \cup R_U) \times L_E)$ *is a set of edges,* $u_0 \in U$ *is the unique initial node,* $U_F \subset U$ *is a set of final nodes,* C *is a set of scenario charts,* $f_U : U \rightarrow$ *is a total, injective function mapping each use case node to a scenario chart and* $m_U : U \cup R_U \rightarrow \{+, -\}$ *is a total function marking whether the use case or region can have multiple concurrent executions.*

A 3-level use case chart is well-formed if all edges map use case nodes to use case nodes or scenario nodes to scenario nodes. In other words, there should be no edge that links a use case node and a scenario node. Formally, $dom(f_S) \cap dom(f_U) = \emptyset$.

3 Use Case Chart Semantics

This section sketches a trace-based semantics for use case charts. A trace semantics is used to achieve consistency with existing semantics for sequence chart notations. A trace is a sequence of events where an event may be a send event, $!x$, or a receive event, $?x$. The semantics of a 3-level use case chart is defined as follows.

Definition 3. *The semantics of a 3-level use case chart,* U*, is a pair of trace sets,* (P_U, N_U)*, where* P_U *is the set of positive traces for* U *and* N_U *is the set of negative traces for* U*.*

Positive traces are traces that are possible in any implementation of the use case chart. Negative traces may never occur in a valid implementation of the use case chart. An implementation satisfies a use case chart if every positive trace is a possible execution path and if no negative trace is a possible execution path.

The details of the semantics are given in stages — first, the semantics of UML2.0 interaction diagrams is given, followed by the semantics for scenario charts, and finally, use case charts.

3.1 Semantics of UML2.0 Interaction Diagrams

A message in a UML2.0 interaction has two events — a send event and a receive event. The send event must come before the receive event. In UML2.0, as shown in Figure 4, messages are composed using interaction fragments, where a fragment has an interaction operator and a number of interaction operands. For example, Figure 4 has two unary-operand fragments — one with the **all** operator and one with the **exist** operator.

The default operator is **seq** which represents weak sequencing. Any messages not explicitly contained within a fragment are by default assumed to be contained within a **seq** fragment. **seq** fragments are defined in UML2.0 to have a weak sequencing semantics ([UML05]):

- The ordering of events within each operand is maintained.
- Events on different lifelines from different operands may come in any order.
- Events on the same lifeline from different operands are ordered such that an event from the first operand comes before an event from the second operand.

Any **seq** fragment joins two traces — one from each of its operands – in a way that satisfies these three constraints. The positive traces for **seq** are all possible ways of joining a positive trace from the first operand and a positive trace from the second operand. The negative traces for **seq** are those derived from joining a positive trace from the first operand with a negative trace from the second, or a negative trace from the first with either a positive or negative trace from the second.

A trace semantics for the other interaction operators can be given in the same way. For example, for **alt**, the set of positive traces is the union of the set of positive traces from each operand. Similarly, the set of negative traces is the union of the set of negative traces from each operand. **par** is defined by interleaving traces from each of its operands. Its positive traces are the interleavings of positive traces from both operands. Its negative traces are the interleavings of negative traces from both operands, or a positive trace from one operand with the negative trace from the other operand. The **neg** operator simply negates all traces — its set of negative traces is the union of the positive and negative traces of its operand. This captures the fact that the negation of a negative trace remains negative. These notions are based on the formalization of UML2.0 interaction diagrams given in [HHRS05].

The semantics for the multiobject extensions are now given. Multiobjects cannot be used unless either an existential or universal operator is also used.

Hence, only the semantics for the operators **exist** and **all** need be given. Suppose that an **all** fragment is applied to a positive trace of events, $t_1, t_2, \ldots$. The resulting positive traces are all those that can be derived by replacing each t_i by its image under **all**. Suppose that t_i is a receive event where the receiving instance is stereotyped as a multiobject. Then the image under **all** is given as the trace $t_{i_1}, t_{i_2}, \ldots$ where each t_{i_j} is the same event but received by instance j. The corresponding send event is also replaced by a set of send events, one for each instance j. The same logic applies if t_i is a send event where the sending instance is a multiobject. In this case, t_i is replaced by a set of send events, one for each instance of the multiobject, and the corresponding receive events for the new send events are added. If, for a message, both its sending and receiving instances are multiobjects, the preceding rules result in duplication which is just removed.

When an **all** fragment is applied to a negative trace of events, the semantics is derived in the same way as in the previous paragraph. For example, a negative send event is replaced with multiple negative send events, one for each instance of the multiobject, and the corresponding negative receive events are added.

The case for **exist** is the same except that positive (alternatively, negative) traces for the **exist** fragment are those in which t_i is replaced by a trace for just one of the multiobject instances, not all of them.

3.2 Semantics of Scenario Charts

As stated in the previous section, the semantics of an interaction diagram is given as a pair of sets of traces. The semantics is extended to scenario charts in the natural way — the semantics of a scenario chart is also given by a set of positive traces and a set of negative traces.

Edges of type *normal* in scenario charts can be given a semantics by "flattening" the edge — i.e., create a new interaction diagram that takes the interaction diagrams represented by the source and target and connects them using an interaction fragment with a particular interaction operator. Normal edges with only one source and target edge are flattened using the **seq** interaction operator for sequential composition. This captures the weak sequential semantics of one-to-one *normal* edges. Many-to-many *normal* edges are flattened using the **par** interaction operator. This is because the semantics of a one-to-many edge is defined to be a forking and that of a many-to-one edge is defined to be a joining of "threads". Hence, a many-to-many edge can be replaced by a fork and join in the usual activity diagram notation. Since *normal* edges can be eliminated in this way, their semantics is not explicitly given here but the semantics is assumed to be that of the equivalent "flattened" interaction diagram. This leaves only edges of type *neg*, *preempts* and *suspends* to be dealt with.

In what follows, c_1 **preempts** c_2 informally means that scenario c_1 preempts c_2. c_1 **suspends** c_2 means that c_1 suspends c_2 and c_1 **negative during** c_2 means that c_1 can never happen during the execution of c_2.

The semantics for preemption, suspension and negation are given only for one-to-one edges, but can be extended in the obvious way to many-to-many

edges: for example, if there is a preemption edge with multiple target nodes, it means that all those nodes are preempted by the source(s) of the preemption edge.

For preemption, a positive trace for c_1 **preempts** c_2 is any trace made up of a prefix of a positive trace of c_2 concatenated with a positive trace of c_1. Note that a preempting scenario cannot have negative traces (if it does, they are simply ignored). Furthermore, c_1 **preempts** c_2 does not introduce any new negative traces because preempting traces have no effect on the original negative traces. The case for suspension is similar except that control returns to the suspended scenario once the suspending scenario is complete.

For the negation case, the positive traces of c_1 **negative during** c_2 are simply the positive traces of c_2. Negative traces, however, can be any trace that is an interleaving of a positive trace of c_2 with a positive trace of c_1. This, in effect, defines a monitor for traces of c_1 — if a c_1 trace occurs at any point, even with events interleaved with its own events, then an error state has been entered. Note that c_1 cannot have negative traces.

The semantics for multiple concurrent executions (the asterisk notation) is given by interleaving and hence can be described in terms of flattening using **par** operators. The number of **par** operators is unbounded since there can be any number of executions of the node.

For the most part, regions are merely syntactic sugar. The exception is when a region is defined as having multiple concurrent executions. In this case, the traces of the region are given by the semantics of multiple executions as described in the previous paragraph.

The semantics for final success and final failure nodes are given in the following subsection.

3.3 Semantics of Use Case Charts

The semantics for use case charts is essentially the same as for scenario charts because both a scenario and a use case are given meaning as a set of traces. For use case charts, however, the meaning of a *normal* edge is given by strong not weak sequential composition. This means that before execution can continue along an edge to the next use case, *all* instances of participating classifiers must complete (where completion is defined below). In contrast, in scenario charts, some instances may complete and continue to the next node while others remain in the current node. Strong composition is chosen to define use case charts because nodes represent use cases. Use cases are considered modular functional units in which the entire unit must complete before control goes elsewhere. Strong composition enforces the modularity and, in most cases, is probably adequate. However, the author acknowledges that, in certain situations, weak composition may be desired and future versions of use case charts may allow the modeler to choose the type of composition. Semantically, strong composition of traces is just concatenation.

A use case chart node completes if and only if its defining scenario chart reaches a final success or final failure node. If the scenario chart reaches a final

success node, control continues to the next use case node. If the scenario chart reaches a final failure node, the use case "thread" terminates. Semantically, each trace in a scenario chart is either infinite, ends with a final success node (a success trace) or a final failure node (a failure trace). Suppose a use case chart has two nodes, U_1 and U_2, connected by a single edge from U_1 to U_2. Then the positive trace set of the use case chart is the union of three trace sets: the positive infinite traces of U_1, the set of traces formed by concatenating positive success traces from U_1 with positive traces from U_2, and the set of positive failure traces from U_1. The extension to negative traces is straightforward.

4 Related Work and Conclusion

A variety of notations for scenario-based definitions exist, such as UML2.0 interaction overview diagrams, high-level message sequence charts (hMSCs) and approaches based on activity diagrams (e.g., [Smi04]). Notationally, the extensions that use case charts provide are relatively minor. The notation is based on UML2.0 activity diagrams and some extensions have been suggested by other authors — e.g., [Kru00] deals with preemption for hMSCs. The contribution is that use case charts are a usable yet precise notation that can be directly executed. Work towards simulating use case charts is currently underway. A state machine-based simulator for interaction diagrams has already been defined [WS00] and a simulator for the remaining parts of use case charts is being developed.

Use case charts are intended to be used as a way to rapidly simulate use case scenarios. As such, they can be used either during requirements engineering or early design. Clearly, the appropriateness of their use depends on the application and they are more suited to complex, reactive systems. For example, so far they have been used to model air traffic control and telecommunication applications. In such applications, the sequence of interactions in use cases quickly becomes very complex and stakeholders quickly question the validity of the interactions they have developed. By using use case charts, these interactions can be simulated either during requirements gathering or early design. The focus on reactive, concurrent and distributed systems means that interaction diagrams are the most suitable notation for the level 3 models although, in principle, there is no reason why level 3 could not be based on activity diagrams as well. Note that use case charts are not meant to be a substitute for use case diagrams. The two diagram types are usually used together — use case diagrams focus on the actors whereas use case charts focus on the interactions (the initiating actors are implicitly defined at level 3).

References

[HHRS05] Øystein Haugen, Knut Eilif Husa, Ragnhild Kobro Runde, and Ketil Stølen. Stairs: Towards formal design with sequence diagrams. *Journal of Software and System Modeling*, 2005. To Appear.

[Kru00] Ingolf Krueger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universitaet Muenchen, 2000.

[SEG] University of Paderborn Software Engineering Group. Shuttle system case study. http://wwwcs.uni-paderborn.de/cs/ag-schaefer/CaseStudies/ShuttleSystem/.

[Smi04] Michal Smialek. Accommodating informality with necessary precision in use case scenarios. In *Proceedings of Workshop on Open Issues in Industrial Use Case Modeling at UML2004*, 2004.

[UML05] Unified modeling language 2.0 specification, 2005. http://www.omg.org.

[WA04] Jon Whittle and Joao Araujo. Scenario modelling with aspects. *IEE Proceedings — Software*, 151(4):157–172, 2004.

[WS00] Jon Whittle and Johann Schumann. Generating statechart designs from scenarios. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 314–323, New York, NY, USA, 2000. ACM Press.