Electronic Notes in Theoretical Computer Science 50 No. 3 (2001) – Proc. GT-VMT 2001
URL: http://www.elsevier.nl/locate/entcs/volume50.html 9 pages

Integrating Model Information in UML Sequence Diagrams

Aliki Tsiolakis [1]

Department of Mathematics and Computer Science
Universität Bremen, Germany

Abstract

In a UML model, different aspects of a system are covered by different types of diagrams. Nevertheless, it is important to provide means to check the consistency and completeness of the UML model. This problem is addressed in this paper by integrating the information specified in class and statechart diagrams into sequence diagrams. The representation as constraints attached to certain locations of the objects lifelines in the sequence diagram allows the identification of gaps and contradictory specifications. Furthermore, dependencies between the sequence diagrams can be investigated based on the previous analysis.

1 Introduction

The Unified Modeling Language ([4]) has become the de-facto standard for object-oriented modelling. Different views of a system are represented by different diagram types. This paper focuses on UML use case, class, statechart, and sequence diagrams. Thus, the system's functionality, its static structure, internal behaviour, and the interactions of parts of the system are considered.

Although the system modellers may benefit from looking at the different views independently, there exists a high risk that the overall specification is inconsistent or incomplete. Incompleteness may occur if necessary information (e.g., about the behaviour of an object) is either absent or specified in another diagram. Using the relationships between the different diagram types as described in the UML standard, it is possible to integrate the properties specified in the system's diagrams. The aim of this paper is to describe an approach how to integrate the information in sequence diagrams and how to check it for consistency and completeness using the additional information. Furthermore, the relationships and dependencies of a collection of sequence diagrams can be represented abstractly in the corresponding use case diagram.

[1] Email: tsio@tzi.uni-bremen.de

The *integration algorithm* can be summarised as follows:

Phase 1: Analysis to retrieve information from different diagrams

Phase 2: Synthesis of context conditions by checking for consistency and completeness

Phase 3: Investigation of the dependencies of a collection of sequence diagrams and their representation

The structure of this paper follows the above algorithm. In the remainder of this section a small example and the UML diagrams used for its specification are introduced.

1.1 Client-Server Example

Throughout the paper, a simple client-server system is used that is based on the Dynamic Host Configuration Protocol (DHCP) ([2]) — a protocol widely used in networks to assign unique network addresses to clients. The main idea is that a client can try to discover servers which can offer it an unused IP address (i.e., a network address that is not assigned to another client). The client chooses one of the offered IP addresses by requesting it and usually gets a positive acknowledgement from the server. A client initiates this procedure to connect to the network and may afterwards disconnect by releasing its IP address explicitly. The example is constructed in a way that each server has exactly one IP address and is allowed to offer it to more than one client, so that only the client that sends the first request gets a positive acknowledgement and the other clients get a negative acknowledgement.
This simple system is modelled using the following diagrams:[2]

(i) A *use case diagram* specifies the client's connection to and disconnection from the network (use case Connect and Disconnect). The procedure is initiated by a user of the system.

(ii) A *class diagram* shows the classes Client, Server and IPAddress. The latter provides an operation getBinding that returns `true` if the particular address has already been assigned to a client and `false` otherwise.
Also, the class diagram can be further refined by *object constraints* attached to the classes. These constraints include invariants for the classes and their attributes, and pre- and postconditions for their operations. It is also possible to represent the multiplicity adornments at the association ends as constraints. The constraints can be given in formal textual notation using the Object Constraint Language (OCL) for expressions. An example for a postcondition of the server's operation getFreeIP() is:

```
context Server::getFreeIP():  IPAddress
  post: result.getBinding() = false
```

[2] Only a few diagrams are shown in this abstract. The complete model is given in [3].

(iii) The realisation of the use cases in terms of cooperating objects can be specified in *sequence diagrams*. In this example, there are two scenarios of use case Connect – one that shows a successful connection to the network (scenario 1) and another one that shows an interaction where the client finally gets a negative acknowledgement (scenario 2).

(iv) Finally, the internal behaviour of the objects of class Server and Client are shown in *statechart diagrams*.

2 Retrieving Information from Different Diagrams

The analysis phase of the algorithm comprises a detailed examination of the properties that are relevant for the overall functionality, but are distributed across different diagrams of different types. These properties include behavioural aspects of individual objects (e.g., the effects of communication and their constraints with respect to different states of the communicating objects) as well as structural aspects (e.g., constraints in form of data and multiplicity invariants and method specifications). The idea presented in this paper is that to integrate the model information the messages in the sequence diagram and their method invocations have to be checked with respect to all properties of the system: The possible states of the communicating objects, the method invocations and guards in statechart diagrams, and the method specifications and data and multiplicity invariants in class diagrams.

The properties will be represented in a sequence diagram as constraints that are attached to the lifelines of the object instances. By enriching the sequence diagram in such a way and thus collecting the distributed properties, they are represented in a way that enables the synthesis. In the following, the generation of these constraints and the locations of their attachment are discussed.

Analysing Statechart Diagrams

The capability of an object to execute some method is specified in a statechart by the states and guards that support the corresponding method invocation event. To consider a message in the sequence diagram, the statecharts of the sender and receiver object have to be investigated. If there is a transition in the sender's statechart diagram that effects the sending of the particular message, the sender object is in the transition's target state after sending the message. If there is a transition in the receiver's statechart diagram that can be triggered by the particular message, the receiver has to be in the transition's source state beforehand. Additionally, the guard has to hold before the message can be received. Thus, constraints containing the OCL expressions of the states and the guards can be attached to the sender's or receiver's lifeline, respectively. The point for each attachment is a location after the message sending or before the message reception, respectively.

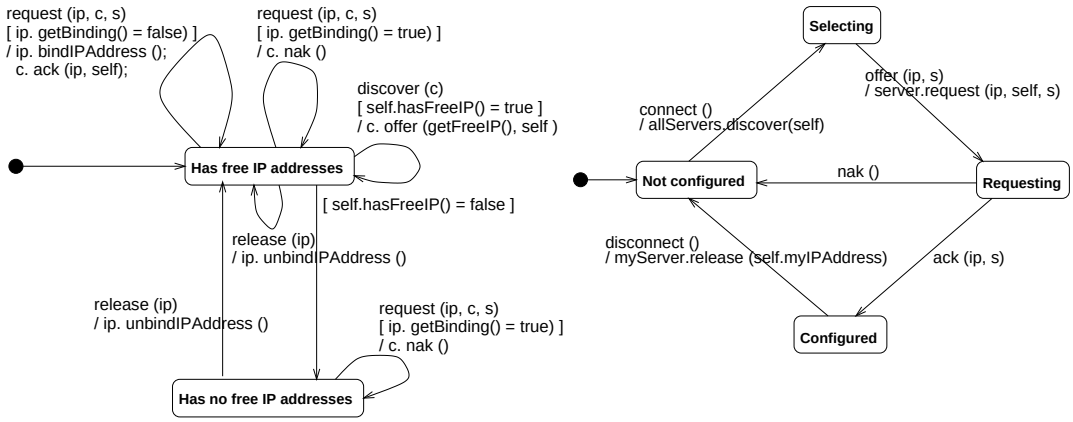

Fig. 1. Statechart diagrams of class Server and class Client

Figure 1 shows the statechart diagrams of class Server and class Client. If, for example, the message request (ip, c, s) of a sequence diagram is analysed, the client object is after the sending in the state Requesting and the server object is beforehand either in state Has free IP addresses or in state Has no free IP addresses. As well, the guard conditions have to hold and constraints containing the guards are attached to the same location on the receiver's lifeline as the state constraint. Since, in this example, there are three possible transitions in the server's statechart diagram the state constraints and guard constraints are labelled with their transitions names and combined by the boolean operator or. By analysing the next message in the sequence diagram (nak ()), the number of previously possible transitions and thus the number of state and guard constraints can be reduced. [3] Figure 2 shows scenario 2 of use case Connect extended with additional constraints (i.e., state and guard constraints) resulting from analysing the statechart diagrams.

Analysing Class Diagrams and Object Constraints

By analysing a class diagram and its separate object constraints with respect to a particular sequence diagram, constraints containing the pre- and postconditions and the data and multiplicity invariants are attached to the lifelines of the objects. Since data and multiplicity invariants have to be maintained at any state of the object, these constraints are attached to all locations of the respective object. [4] To find out the correct locations of the pre- and postconditions, each message of the particular sequence diagram has to be examined. When considering a message, the constraints containing the pre- and postconditions of the operation that is invoked by the message are

[3] Messages that are initiated as an action of the previous message' transition reduce the number of previously possible transitions because only those transitions are relevant that have this method invocation in their sequence of action.

[4] The algorithm restricts the attachment of constraints to those locations where a state constraint has already been attached. Thereby it is avoided to check any constraint in unstable intermediate states.

attached to the receiver's lifeline. The constraint containing the precondition is placed at the location immediately before the message reception on the respective object's lifeline, the constraint containing the postcondition it located after the message reception.

Regarding the message getFreeIP () in Figure 2, the corresponding postcondition (as described on page 2) has to be attached to the server's lifeline at the location after the reception of the message (location s2). Figure 2 shows scenario 2 of use case Connect extended with the additional constraints containing the states, guards, data or multiplicity invariants, and pre- and postconditions.

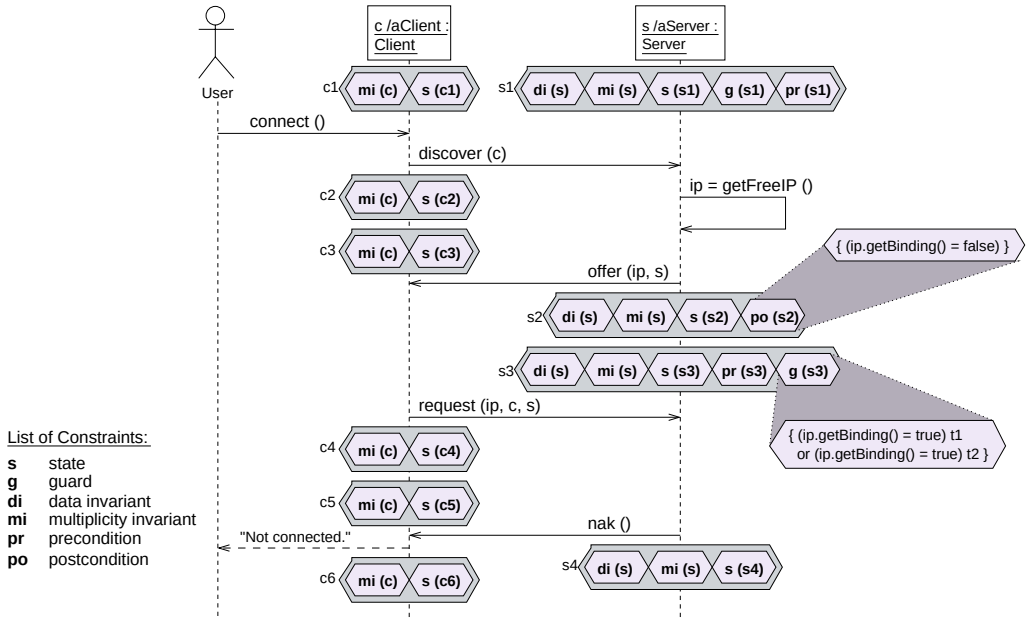

Fig. 2. Sequence diagram that shows scenario 2 of use case Connect with additional constraints containing the properties of the statechart diagrams and the class diagram. The locations with attached constraints are labelled (e.g., s2).

3 Synthesising Context Conditions in Sequence Diagrams

During the analysis phase, the sequence diagrams of a system have been enriched with additional constraints containing the properties of the statechart diagrams and the class diagram. In the synthesis phase, these context conditions are examined to identify constraints that are contradictory. These **critical points** are marked in the sequence diagram.

First of all, the context conditions attached to one of the locations are investigated (*horizontal synthesis*). Since each of the constraints and as well their combination[5] is a logical expression, they are checked together. If the

[5] Constraints of different origin are always combined by the and-operator.

constraints can be evaluated to true there exists no inconsistency, otherwise the model is inconsistent or incomplete. The result may depend on the values of variables, e.g., the values of attributes. If for each possible assignment of the variables the logical expression is evaluated to false, it is contradictory and definitely points out a critical point.

Afterwards, the constraints at two succeeding locations on the same lifeline are checked (*vertical synthesis*). That means, the following combinations of locations have to be investigated for the sequence diagram in Figure 2: (c2, c3), (c4, c5), and (s2, s3). If a combination leads to a contradiction either some additional behaviour must happen in between that is not shown in the sequence diagram, or the sequence of messages is not consistent with the behaviour described in the statechart diagram of the object.

In Figure 2, the constraints at (s2, s3) expose a contradictory specification: the constraint containing the postcondition at location s2 (po(s2)) contradicts the guard constraint at location s3 (g(s3)). This yields a critical point in between these locations, indicating in this case a dependency of another scenario (cf. Section 4).

Finally, it is possible to generate **pre- and postconditions for each sequence diagram** using the attached constraints. The precondition is generated using the constraints attached to the first location of each object's lifeline (e.g., the combination of the constraints at location c1 and s1). The postcondition contains the constraints attached to the last locations (e.g., the constraint at location c6 and s4). The resulting constraints are attached to the lifelines directly after the object symbols (i.e., before the sending or receiving of any message) and to a new location after the last locations, respectively. The pre- and postconditions contain the object configurations of the sequence diagram and the possible states of the objects (i.e., the states of the objects, and possible attribute values) before and after the specified scenario, respectively.

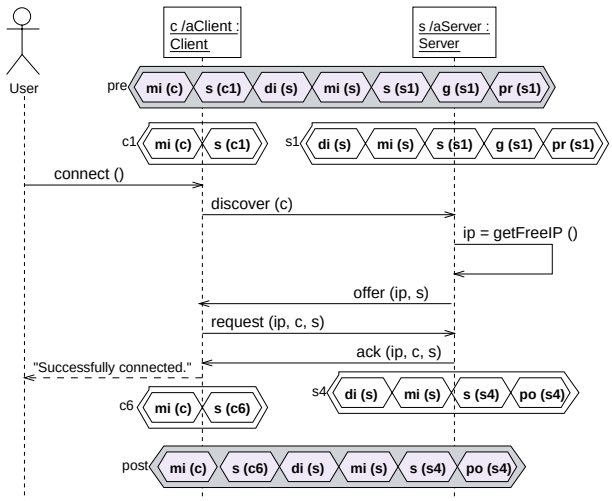

Fig. 3. Sequence diagram with pre- and postconditions

Figure 3 shows the pre- and postconditions for the sequence diagram that describes scenario 1 of use case Connect. The pre- and postconditions are attached to the new locations labelled pre and post.

4 Dependencies between Sequence Diagrams

Considering a collection of extended sequence diagrams as described above, the pre- and postconditions can be used to refine some of the use cases, to identify possible "inclusions" of sequence diagrams at critical points, and to define a required chronological sequence.

Refinement of Use Cases

If a use case is realised by more than one scenario (i.e., by more than one sequence diagram), it can be further refined. Therefore, a new use case for each scenario is specified that extends the base use case.

"Sequence Diagram Inclusion"

Since one of the explanations of a critical point in a sequence diagram can be the occurrence of additional behaviour between the two contradictory constraints, the collection of sequence diagrams can be investigated to identify sequence diagrams that can be included at that location. Therefore, the combination of the constraints at the critical points and the pre- and postconditions of the other sequence diagrams in the collection are considered. If there exists a sequence diagram such that the respective constraints conform, the behaviour of this sequence diagram could have happened in between. Thus, it is possible to include this sequence diagram at the identified location.

Figure 4 shows the possible inclusion of a sequence diagram at the critical point of the sequence diagram that describes scenario 2 (see Figure 2). The behaviour of the included sequence diagram is shown in Figure 3 and realises the first scenario of use case Connect.

Temporal Order of Sequence Diagrams

Analysing the pre- and postconditions of a collection of sequence diagrams it is possible to derive a temporal order of these diagrams. This is derived from the causal relationships detected in the analysis and means that some previous behaviour defined in one sequence diagram is required before the application of another one. As well, it is specified if the application of some sequence diagrams is independent. For example, a sequence diagram with an empty precondition (i.e., all objects are created during its application) shows that no previous interactions are required. Generally, the application of a sequence diagram may depend on the previous application of another one if the pre- and postconditions overlap, respectively.

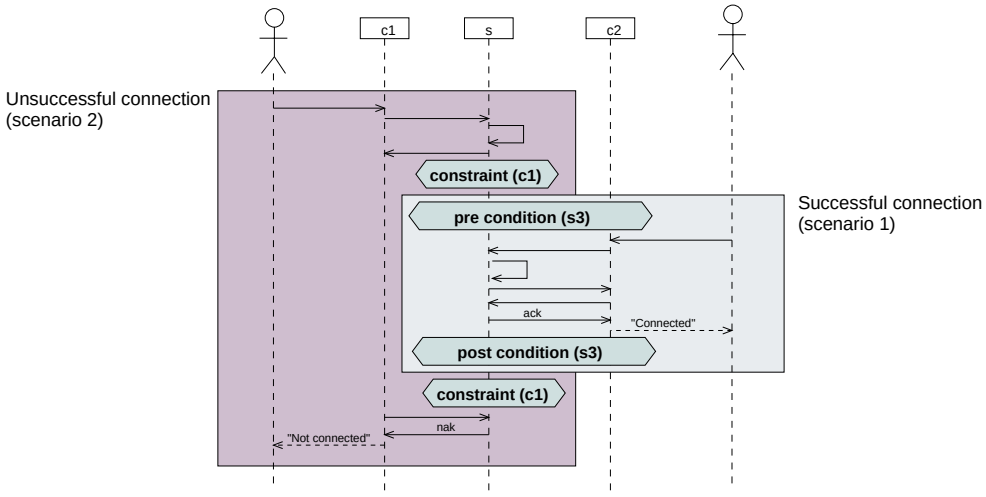

Fig. 4. Sequence diagram inclusion at the critical point of a sequence diagram.

For example, the postcondition of the sequence diagram in Figure 3 (use case Connect) specifies that at least a client and a server object exist and that the client is in the state Configured. The precondition of the sequence diagram that realises the use case Disconnect defines a compatible object configuration and compatible states of the objects. Thus, this sequence diagram requires the previous application of the sequence diagram in Figure 3.

For a representation of these dependencies it is suggested to use predefined relationships between use cases (e.g., associations with the stereotype `<<include>>` or `<<extend>>`) or to define suitable new stereotypes (e.g., `<<require>>` or `<<refine>>`). This is worked out in detail in [3].

5 Conclusion

To reduce the risks and expenses for debugging the implementation of a system, it is highly advisable to integrate a semantic analysis of the UML diagrams into the design phase of the development process. This paper describes an approach to integrate the "distributed" information specified in UML class and statechart diagrams into the sequence diagrams and thereby to identify contradictory or incomplete specifications and dependencies of the sequence diagram scenarios. The latter can then be abstracted and included into the use case diagram to enhance the model.

For a consequent and easy support of the development process by this approach, tool support is necessary. Thus, directions of future work might include the development of a UML modelling tool that implies the automatic integration of the different views and their consistency checking. Since the identification of possible contradictions by examining the logical expressions of the constraints is a main part of this checking algorithm, the integration of a theorem prover should be envisaged.

Tool-supported software development necessitates precise definitions of

these concepts. A formalisation of (extended) sequence diagrams based on the formalism of Life Sequence Charts ([1]) is defined in [3]. This definition includes the abstract syntax of these sequence diagrams and should be expanded to specify also the semantics of the algorithm.

References

[1] W. Damm and D. Harel. "LSCs: Breathing Life into Message Sequence Charts". Technical report, July 2000. Revised version.

[2] R. Droms. "Dynamic Host Configuration Protocol (DHCP)". IETF, Networking Group, March 1997. Available at `http://www.ietf.org/rfc/rfc2131.txt` .

[3] A. Tsiolakis. "Semantic Analysis and Consistency Checking of UML Sequence diagrams". Technical report No. 2001-06, Technische Universität Berlin. April 2001. Available at `http://www.cs.tu-berlin.de/~aliki/`.

[4] Object Management Group. "OMG Unified Modeling Language, Specification – version 1.3", March 2000. Available at `http://www.omg.org/uml/` .