Refining UML interactions

Ragnhild Kobro Runde
Department of Informatics, University of Oslo, Norway

Abstract

STAIRS is an approach to the compositional development of interactions, supporting the specification of mandatory as well as potential behaviour. The classical notion of refinement is supported both by using a formal denotational trace semantics and by more practical transformation rules together with pragmatic constraints.

Introduction

STAIRS [HS03] is an approach to the compositional development of UML interactions, such as sequence diagrams and interaction overview diagrams. The process of developing the interactions is viewed as a process of learning through describing. From a fuzzy, rough sketch, the aim is to reach a precise and detailed description applicable for formal handling.

An essential feature in STAIRS is the ability to distinguish between mandatory and potential behaviour. By potential behaviour, we mean that a specification may give several alternative behaviours serving the same overall purpose, and that fulfilling only some of them is acceptable for an implementation to be correct. On the other hand, mandatory behaviours are alternatives that must all be present in a valid implementation.

Figure 1: Mandatory (xalt) and potential (alt) alternatives
As an example, consider the interaction overview diagram in figure 1, specifying the behaviour of an ATM (Automatic Teller Machine). An ATM offers the customer withdrawal of money or the purchase of a number of foreign currencies (in addition to having cash refill). The \texttt{alt}-operator is used to specify that for an implementation of the ATM it is optional to offer either euros or US dollars, or both. We also want to specify that any ATM \textit{must} offer both withdrawal of native money and at least one foreign currency. These mandatory alternatives are expressed using the novel operator \texttt{xalt}, introduced in [HS03].

\section*{Semantics of interactions}

To explain the meaning of interactions, STAIRS uses denotational trace semantics. A trace is a sequence of input/output events, representing one execution of the specified system. In general, an interaction is viewed as specifying a set of positive and/or negative traces. Traces not specified as either positive or negative are called inconclusive.

A central notion in STAIRS is that of an interaction obligation, used to capture mandatory behaviour. An interaction obligation is a pair \((p, n)\) of sets of traces where the first set is interpreted as the set of positive traces and the second set is the set of negative traces.

An interaction is given its semantics as a set of interaction obligations. Any valid implementation of the interaction must contain at least one positive trace from each of the interaction obligations. Different interaction obligations are specified syntactically by using the novel operator \texttt{xalt}, where each operand gives rise to one obligation. The positive traces within one obligation are alternatives that \textit{may} be present in an implementation, thus supporting specification of potential behaviour. (At the specification level this is, among other things, achieved by the \texttt{alt}-operator.)

Formally, the semantics of interactions is defined by a function that for any interaction \(d\) yields a set \(\llbracket d \rrbracket\) of interaction obligations. For instance, we have

\[
\begin{align*}
\llbracket d_1 \texttt{alt} d_2 \rrbracket & \overset{\text{def}}{=} \{ (p_1 \cup p_2, n_1 \cup n_2) \mid (p_1, n_1) \in \llbracket d_1 \rrbracket \wedge (p_2, n_2) \in \llbracket d_2 \rrbracket \} \\
\llbracket d_1 \texttt{xalt} d_2 \rrbracket & \overset{\text{def}}{=} \llbracket d_1 \rrbracket \cup \llbracket d_2 \rrbracket
\end{align*}
\]

For more details, see [HHRS04].

\section*{Refinement of interactions}

An important part of STAIRS is to explain the classical notion of refinement in the setting of interactions. This is done by distinguishing between three main kinds of system development steps: supplementing, narrowing, and detailing. Supplementing categorizes inconclusive behaviour as either positive or negative, while narrowing reduces the set of positive behaviours. Detailing means to introduce more detailed descriptions, while maintaining the essential behaviour.
Formally, we have that an interaction obligation \((p_2, n_2)\) is a refinement of an interaction obligation \((p_1, n_1)\), written \((p_1, n_1) \leadsto_r (p_2, n_2)\), iff \(n_1 \subseteq n_2 \land p_1 \subseteq p_2 \cup n_2\). An interaction \(d'\) is a refinement of an interaction \(d\) iff \(\forall o \in \{d\} : \exists o' \in \{d'\} : o \leadsto_r o'\).

Supplementing and narrowing are special cases of the general notion of refinement. An interaction obligation \((p_2, n_2)\) supplements an interaction obligation \((p_1, n_1)\) iff \((n_1 \subseteq n_2 \land p_1 \subseteq p_2) \lor (n_1 \subseteq n_2 \land p_1 \subseteq p_2)\). An interaction obligation \((p_2, n_2)\) narrows an interaction obligation \((p_1, n_1)\) iff \(p_2 \subseteq p_1 \land n_2 = n_1 \cup (p_1 \setminus p_2)\).

Detailing corresponds to interface refinement as defined in e.g. Focus [BS01].

In [HHRS04] we prove that refinement as defined above is transitive and monotonic with respect to the most common interaction operators. Transitivity is important, as it ensures that successive refinement steps results in a valid refinement of the original specification. In the same way, monotonicity allows for the different parts of an interaction to be refined independently.

**Refinement calculus**

The semantic details as presented in [HHRS04] are quite involved, and it is unlikely that most users of interactions will be interested in working at that level. Thus, there is a need for a more practical, high-level approach to dealing with refinement of interactions. We are currently working on constructing a refinement calculus for STAIRS. In this first phase, the calculus consists of two kinds of rules:

**Transformation rules** describing syntactical modifications on interactions, modifications that will result in valid refinements.

**Pragmatic constraints** to assist in developing useful refinements, not only valid ones. (For instance, it is very easy to refine a consistent specification into an inconsistent one, thus making it not implementable.)

**References**

