Implementation of extension and reduction relations for incremental development of behavioural models

Hong-Viet Luong, Thomas Lambolais, Anne-Lise Courbis

Implementation of Extension and Reduction Relations for Incremental Development of Behavioural Models

Hong-Viet Luong, Thomas Lambolais, Anne-Lise Courbis
Laboratoire LGI2P, École des Mines d'Alès
Site EERIE, Parc Scientifique Georges Besse
30 035 Nîmes cedex 1, France
{Hong-Viet.Luong, Thomas.Lambolais, Anne-Lise.Courbis}@ema.fr

Abstract

We present a way to implement refinement relations over transition systems, useful for incremental model development. The extension relation, which preserves already modelled functionalities, is suited for refinements. The calculability of this relation relies on the generalization of a bisimulation relation applied on acceptance graphs. It is formally demonstrated and illustrated through an example whose analysis is performed by a Java prototype we have developed. This method can be adapted to UML state machines, which lack evaluation means.

1 Introduction

Our area of interest concerns the build-up of UML behavioural models (state machines) according to an incremental approach. Such an approach is natural as it is based on a widely held mental process because, on the one hand, specifying a complex system requires a hierarchical and iterative approach, and on the other hand, initial specification cannot be considered as complete and must be updated all along the modelling process. The problem we address is to compare a model obtained at a given stage with models derived from previous stages: how is it possible to guarantee that updates do not introduce inconsistencies and preserve already modelled functionalities?

Little work has dealt with this problem in the context of UML modelling, despite the fact that the UML standard [1] is increasingly used in the industry. There are some UML simulation frameworks supporting behavioural model observation, but such intuitive and experience-based evaluations cannot be considered as systematic and reliable. Our goal is therefore to develop formal comparison techniques and tools ensuring consistency between behavioural models, taking into account the distinctive features of incremental modelling. When we ask industrialists questions about their experience, their know-how not to say their methodology in state machines modelling, answers can be summarized as follows. For the most part, models are repetitively custom-written from scratch, there is no capitalization of modelling knowledge or customized library which could offer templates of state machines to be completed, configured and reused. Models are built up in several stages to meet some requirements considered as the most important, other ones being left out at the early phase of the process. When draft models are set up, they are refined by an iterative process until they meet requirements. During the incremental process, indeterminism is introduced to face problems of reasoning or to make modelling easier. Reducing or adding indeterminism may occur at any stage of the process. It is difficult to add new behaviours to a state machine without affecting already modelled functionalities. It means that modellers can face problems of abstraction, refinement, verification, extension and indeterminism. They are lacking adequate tools to support the incremental modelling process.

As these issues are not much addressed in the literature [2, 3], our first approach is to study tried and tested solutions of domains closely related to state machine modelling. Solutions have been found in works dealing with Labelled Transition Systems (LTS) and precisely on may, must [4] and testing preorders, as well as conformance, extension and reduction relations. Two of these criteria, reduction and extension, appear to be specifically interesting in the context of incremental modelling, but their calculability has not yet been established. Our first goal is therefore to demonstrate it. The proof is based on bisimulation relation applied on transformed graphs, as it has been done for *may* and *must* relations [5]. Applying this result on UML state machines [6, 7] is not trivial and requires a long-term analysis, especially to deal with the complete standard. However, we illustrate the extension relation on a case study modelled

both in LTS and its corresponding simplified UML model.

The paper is organized in three parts. At first, we present main definitions useful to understand studied relations and other parts of the paper. We give an overview of these relations and compare them according to different criteria in order to underline the benefits of reduction and extension relations. The second part focuses on the reduction and extension calculability. For this purpose, we give the definition of bisimulation and acceptance graphs by reformulating existing works. In part three, we present the demonstrator developed in JAVA allowing extension relation to be implemented and results obtained on a case study. Lastly, we make a conclusion and present our future works.

2 Analysis of existing relations

In this part, we give fundamental definitions about Labelled Transition System (LTS) [8] and refusal sets [9] to understand the relations we have studied. We present the concept of acceptance set [10] and we demonstrate that the inclusion of refusal sets can be expressed in terms of acceptance sets inclusion. We analyze existing relations allowing models to be compared at the different stages of the modelling and justify our selection for the reduction relation.

2.1 Definitions of conformance relations

A LTS [8] is a graph consisting of states linked by labelled transitions. It models behavioural specifications as well as implementations.

Definition 1. *A LTS $P = (S, Act, \rightarrow, s_0)$ is a tuple consisting of:*

– a non-empty finite set S of states;
– a set Act of actions;
– a transition relation $\rightarrow \subseteq S \times Act \times S$;
– an initial state $s_0 \in S$.

$Act = L \cup \{\tau\}$ where τ represents any internal, unobservable actions, and L is the set of observable actions.

Before presenting the definitions of conformance relation, we give some usual notations:

$$
\begin{aligned}
s \xrightarrow{a} s' &=_{def} (s, a, s') \in \rightarrow \\
s \xrightarrow{a_1..a_2} s' &=_{def} \exists s_0, .., s_n : s = s_0 \xrightarrow{a_1} .. \xrightarrow{a_n} s_n = s' \\
s \xrightarrow{a_1..a_2} &=_{def} \exists s' s \xrightarrow{a_1..a_n} s' \\
s \stackrel{\epsilon}{\Longrightarrow} s' &=_{def} s = s' \text{or } s \xrightarrow{\tau..\tau} s' \\
s \stackrel{a}{\Longrightarrow} s' &=_{def} \exists s_1, s_2 : s \stackrel{\epsilon}{\Longrightarrow} s_1 \xrightarrow{a} s_2 \stackrel{\epsilon}{\Longrightarrow} s' \\
s \stackrel{a_1..a_2}{\Longrightarrow} s' &=_{def} \exists s_0, .., s_n : s = s_0 \stackrel{a_1}{\Longrightarrow} .. \stackrel{a_n}{\Longrightarrow} s_n = s' \\
s \stackrel{\sigma}{\Longrightarrow} &=_{def} \exists s' : s \stackrel{\sigma}{\Longrightarrow} s' \\
s \text{ after } \sigma &=_{def} \{s' | s \stackrel{\sigma}{\Longrightarrow} s'\}
\end{aligned}
$$

$$
\begin{aligned}
P \text{ after } \sigma &=_{def} s_0 \text{ after } \sigma \\
\text{Traces : } Tr(P) &=_{def} \{\sigma \in L^* | s_0 \stackrel{\sigma}{\Longrightarrow}\} \\
Out(P, \sigma) &=_{def} \{a \in L | \sigma.a \in Tr(P)\} \\
D(s, a) &=_{def} \{s' | s \xrightarrow{a} s'\}
\end{aligned}
$$

Definition 2. *Refusal set.*
$Ref(P, \sigma)$ is the refusal set of P after trace σ

$$Ref(P, \sigma) =_{def} \left\{ X | \exists p \in P \text{ after } \sigma . p \stackrel{e}{\not\Rightarrow}, \forall e \in X \right\}$$

The refusal set is a set $Ref(P, \sigma) \subset \mathcal{P}(L)$. If $\sigma \notin Tr(P)$, $Ref(P, \sigma) = \varnothing$.

The conformance relation is defined in the following way:

Definition 3. *Conformance relation* **conf**.
Let P and Q be two LTS,
Q **conf** P *if* $\forall \sigma \in Tr(P), Ref(Q, \sigma) \subseteq Ref(P, \sigma)$.

By preserving the conformance, we can define the reduction and the extension.

Definition 4. *Reduction relation* **red**.
Let P and Q be two LTS,
Q **red** P *if* $Tr(Q) \subseteq Tr(P)$ *and* Q **conf** P.

Definition 5. *Extension relation* **ext**.
Let P and Q be two LTS,
Q **ext** P *if* $Tr(P) \subseteq Tr(Q)$ *and* Q **conf** P.

The relation **conf** is not a preorder relation: **conf** has not the transitivity property. But **red** and **ext** are reflexive and transitive.

2.2 Acceptance set

In this section we present a definition of acceptance sets and we demonstrate that there exists an equivalence relation between refusal sets inclusion and acceptance sets inclusion. This notion will be used in the next section to build up acceptance graphs associated to LTS.

Definition 6. *Acceptance set:*

$$Acc(P, \sigma) = \{X | \exists p' \in P \text{ after } \sigma . X = Out(p', \epsilon))\}$$

The acceptance set represents the "sets of possible actions" of a process after a trace. Intuitively, the inclusion of acceptance set allows us to check whether a process is more deterministic than another.

Definition 7. *Set of sets inclusion*
Let $A, B \subseteq 2^{Act}$. $A \subset\subset B$ if:

$$\forall S \in A.\ \exists S' \in B.\ S' \subseteq S.$$

Theorem 1. $\forall \sigma \in Tr(Q).\ Acc(P,\sigma) \subset\subset Acc(Q,\sigma) \Leftrightarrow Ref(P,\sigma) \subseteq Ref(Q,\sigma)$

Proof. Firstly, we must show the relationship between the inclusion of refusal sets and the inclusion of acceptance sets.

We can rewrite the definition 2 as follows:

$$Ref(P,\sigma) =_{def} \left\{ X | \exists p'.\ P \stackrel{\sigma}{\Longrightarrow} p' \wedge p' \stackrel{e}{\not\Rightarrow}, \forall e \in X \right\}$$

$$\Leftrightarrow Ref(P,\sigma) =_{def} \left\{ X | \exists p'.\ P \stackrel{\sigma}{\Longrightarrow} p' \wedge X \subseteq L - Out(p',\epsilon) \right\}$$

Secondly, we can reformulate the definition 7 by applying the definition 6:

$$\begin{aligned}
&\forall \sigma \in Tr(P).\ Acc(P,\sigma) \subset\subset Acc(Q,\sigma) \\
\Leftrightarrow &\forall \sigma \in Tr(P).\ \forall X \in Acc(P,\sigma).\ \exists Y \in Acc(Q,\sigma).\ Y \subseteq X \\
\Leftrightarrow &\forall \sigma \in Tr(P).\ \forall X \in \{Out(p',\epsilon) | p \stackrel{\sigma}{\Longrightarrow} p'\}. \\
&\exists Y \in \{Out(q',\epsilon) | q \stackrel{\sigma}{\Longrightarrow} q'\}.\ Y \subseteq X
\end{aligned}$$

Let X, Y, Z be three sets, we have: $X, Y \subseteq Z \wedge Y \subseteq X \Leftrightarrow Z - X \subseteq Z - Y$

With $Out(p',\epsilon)$ and $Out(q',\epsilon) \subseteq L$, we have therefore:

$$\begin{aligned}
&\forall \sigma \in Tr(P). \\
&\{(L - Out(p',\epsilon)) | p \stackrel{\sigma}{\Rightarrow} p'\} \subseteq \{(L - Out(q',\epsilon)) | q \stackrel{\sigma}{\Rightarrow} q'\} \\
\Leftrightarrow &\forall \sigma \in Tr(P).\ Ref(P,\sigma) \subseteq Ref(Q,\sigma))
\end{aligned}$$

So we have
$Acc(P,\sigma) \subset\subset Acc(Q,\sigma) \Leftrightarrow Ref(P,\sigma) \subseteq (Q,\sigma)$.

□

2.3 State of the art of relations to compare behavioural models

In this part, we propose a summary of existing relations developed for analyzing LTS in order to get an informal interpretation and a way to compare them and intuitively point out how they can be used in a framework of incremental modelling. Studied relations are **conf**, **red**, **ext** defined by [11, 12] and may, must, testing, defined by [5]. Our analysis is based on the comparison of the sets of actions on which the relations are operating. These sets are:

- *must-do*, representing the set of actions that the system has to do, and consequently cannot refuse;
- *must-refuse*, representing the set of actions that the system has to refuse and consequently cannot do;
- *can-do/can-refuse*, representing the set of actions that the system can do but can also refuse. Such a set exists if the system behaves indeterministically.

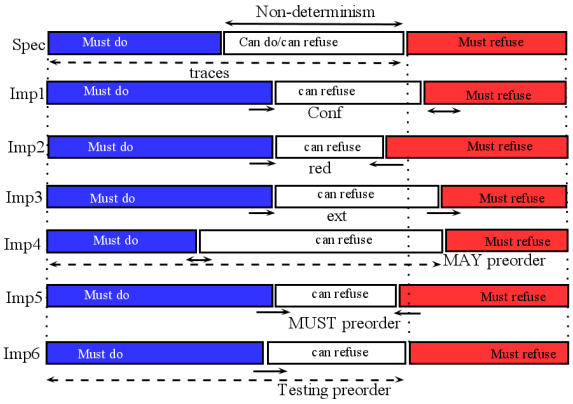

Figure 1. Comparison of studied relations

Note that:

- *may-do* consists of *must-do* and *can-do/can-refuse* sets, which represents the traces of the system;
- *may-refuse* consists of *must-refuse* and *can-do/can-refuse* sets, which corresponds to the refusal sets.

These sets are graphically represented by separate shapes (see figure 1). *Spec* represents the initial model and we study, depending on the applied relation, how the sets change in the new model. The change is indicated by the direction of the arrow under every set shape and means that the set is reduced or extended.

The relation **conf** (see fig. 1 – Imp 1) means that the *must-do* set is extended: it allows the *can-do/can-refuse* and *must-refuse* sets to be reduced or extended. Therefore, this relation guarantees that the new model performs at least as the initial one and may specify new actions. It also guarantees that the new model is as deterministic as the initial one, or more deterministic.

The **red** preorder (see fig. 1 – Imp 2) has the same properties as the relation **conf** but the set of traces is reduced. It thus surely guarantees that the new model is more deterministic that the initial one.

The **ext** preorder (see fig. 1 – Imp 3) has the same properties as the relation **conf** but the set of traces is extended. This relation is therefore interesting for improving models during the incremental modelling process.

The *may preorder* (see fig. 1 – Imp 4) guarantees that the set of traces of the new model contains at least the set of traces of the initial model. Unfortunately, this relation does not verify that the new model is more deterministic since it does not guarantee that the set *can-do/can-refuse* has been reduced. Consequently, the preorder may has to be carefully used during an incremental modelling process.

The *must preorder* (see fig. 1 – Imp 5) is stronger than the relation **conf** since it guarantees that the sets *must-do*

Properties / Relation	Reduction of indeterminism	Addition of actions	Conservation of actions	Suppression of actions
conf	⊛	×	×	×
red	⊛	⊘	×	⊛
ext	⊛	⊛	×	⊘
may	⊘	⊛	×	⊘
must	⊛	⊘	×	⊛
testing	⊛	⊘	⊛	⊘

Symbols: ×: May be supported, ⊛: Guaranteed, ⊘ : Non supported.

Table 1. Overview of studied relations

and *must-refuse* of the new model contain the same set of the initial model. This relation, defined by Hennessy [5], is very similar to the relation **red**, but considers also a notion of divergence over processes.

Finally, *the testing preorder* (see fig. 1 – Imp 6) guarantees that the set *must-do* can be extended without changing the set of traces of the initial model. Consequently, as it was the case for the relation **red** and the must preorder, test surely guarantees that the new model is more deterministic than the initial one.

Table 1 gives a summary of these relations according to four criteria: reduction of indeterminism, extension or reduction of the set of actions, and conservation of traces. For every relation, we identify three cases: it surely guarantees the criteria, it may verify the criteria or it does not deal with the criteria.

The **ext** relation is adequate for incremental developments since it supports refinement approaches. It has the following interesting property:

$$S_2 \ \mathbf{ext} \ S_1 \Rightarrow (\forall I.\ I \ \mathbf{conf} \ S_2 \Rightarrow I \ \mathbf{conf} \ S_1)$$

If we consider **conf** like an implementation relation, it means that all implementations of a refined model is an implementation of the former.

In the next section, we focus on implemetation techniques for this extension relation.

3 Calculability of extension and reduction relations

After having selected the more appropriated relations to compare models and introduced main definitions of the domain, we outline the problem of calculability of the relations **ext** and **red**. As far as we know, these relations have not been implemented yet. Nevertheless, Cleaveland and Hennessy [5] have introduced the concept of acceptance graphs to implement may and must relations. We will follow the same approach to demonstrate the extension and reduction relations. At first, we introduce main definitions about bisimulation and acceptance graphs and next, we present our demonstration.

3.1 Bisimulation

We give a generalisation of the definition of bisimulation given by Milner [8].

Definition 8. *Bisimulation relation.*
Let $\Pi \subseteq S \times S$ and $\Psi_1, \Psi_2 \subseteq S \times Act$. The relation $\mathcal{R}$ $\langle \Pi, \Psi_1, \Psi_2 \rangle$ is a bisimulation if $\mathcal{R} \subseteq \Pi$ and, for all $p, q \in S$, $p\mathcal{R}q$ imply:

1. $\langle p, a \rangle \in \Psi_1 \Rightarrow (p \xrightarrow{a} p' \Rightarrow \exists q'.q \xrightarrow{a} q' \wedge p'\mathcal{R}q')$
2. $\langle q, a \rangle \in \Psi_2 \Rightarrow (q \xrightarrow{a} q' \Rightarrow \exists p'.p \xrightarrow{a} p' \wedge p'\mathcal{R}q')$

When $\Pi = S \times S$ and $\Psi_1, \Psi_2 = S \times Act$, the formula is the same as the bisimulation defined by Milner.

Definition 9. *Largest bisimulation.*
$P \overset{\Pi}{\lessapprox}_{\langle \Psi_1, \Psi_2 \rangle} Q$ if it exists a bisimulation $\mathcal{R}$ $\langle \Pi, \Psi_1, \Psi_2 \rangle$ with $p\mathcal{R}q$.

In this definition, if Ψ_2 is replaced by $\emptyset$, the bisimulation becomes the simulation preorder. The advantage of this definition is to encapsulate the bisimulation and the binary relation Π.

3.2 Acceptance graphs

Before presenting acceptance graphs, we recall the definition of ϵ-closure.

Definition 10. *ϵ-closure of a set of states Q :*

$$Q^\epsilon = \{p | \exists q \in Q.q \overset{\epsilon}{\Longrightarrow} p\}$$

Definition 11. *Acceptance graph.*
$\mathcal{A}(P) = \langle T, Act, \rightarrow_T, t_0 \rangle$ of LTS P is a tuple where:

1. *T is the set of states. $T = \{Q \in 2^S | Q = Q^\epsilon\}$;*
2. *$\rightarrow_T$ is the set of transitions;*
3. *For $t \in T$, we define the acceptance set $t.acc = \{X | X = Out(q, \epsilon) \wedge q \in Q^\epsilon\}$;*
4. *For $A \in t_1.acc$, $a \in A \Rightarrow \exists t_2 \in T$ such that $t_1 \xrightarrow{a} t_2$;*
5. *$t_0 = (\{p_0\})^\epsilon$.*

This definition is similar to acceptance graphs of [5], but the definition of acceptance sets does not take into account divergence states.

Definition 12. $min(A) = \{S \in A |\ \nexists S' \in A.\ S' \subset S\}$.

Lemma 1. $A \subset\subset B \Leftrightarrow min(A) \subset\subset min(B)$.

The algorithm of acceptance graphs construction introduced by [5] can be adapted to the construction of acceptance graphs as defined in this paper.

3.3 Demonstration of the extension and reduction calculability

Theorem 2. *Let P be a LTS and $\mathcal{A}(P)$ its acceptance graph: $Tr(\mathcal{A}(P)) = Tr(P)$.*

Proof. We can prove it by induction with each trace $\sigma \in Tr(P)$ □

Theorem 3. *Let $\Pi = \{\langle t, u\rangle | u.acc \subset\subset t.acc\}$ and P, Q be two LTS:*

1. Q **red** $P \Leftrightarrow \mathcal{A}(P) \subsetneqq^{\Pi}_{\langle\emptyset,\Psi_2\rangle} \mathcal{A}(Q)$
2. Q **ext** $P \Leftrightarrow \mathcal{A}(P) \subsetneqq^{\Pi}_{\langle\Psi_1,\emptyset\rangle} \mathcal{A}(Q)$

Proof. We are going to prove (2.). (1.) is similar and will not be expressed in this article.

1.) We must prove $\mathcal{A}(P) \subsetneqq^{\Pi}_{\langle\Psi_1,\emptyset\rangle} \mathcal{A}(Q) \Rightarrow Q$ **ext** P

First, we establish the relation $\mathcal{R}\langle\Pi, \Psi_1, \emptyset\rangle$ as follows:

$$\mathcal{R} = \{\forall t' \in \mathcal{A}(P).t \xrightarrow{a} t' \Rightarrow (\exists u' \in \mathcal{A}(Q).u \xrightarrow{a} u' \wedge t'\mathcal{R}u') \wedge (\ u'.acc \subset\subset t'.acc)\}$$

By using the theorem (2), the first part of the definition above can be written:

$$\begin{aligned} &a \in Tr(\mathcal{A}(P)) \Rightarrow a \in Tr(\mathcal{A}(Q)) \\ \Rightarrow &Tr(\mathcal{A}(P)) \subseteq Tr(\mathcal{A}(Q)) \Rightarrow Tr(P) \subseteq Tr(Q). \end{aligned}$$

This is the first condition of the definition of the relation **ext**.

By using the definition of acceptance graph $u'.acc = Acc(Q, s)$ and theorem 1 , the second part of the relation $\mathcal{R}$ can be written:

$$\begin{aligned} &\forall s \in Tr(P).\ Acc(Q, s) \subset\subset Acc(P, s) \\ \Rightarrow &\forall s \in Tr(P).\ Ref(Q, s) \subseteq Ref(P, s) \end{aligned}$$

We have therefore the conformance.

2.) We prove ($\Rightarrow$)

Suppose that Q **ext** P. We must prove the relation $\mathcal{R}\langle\Pi, \Psi_1, \emptyset\rangle$ as defined above.

Q **ext** $P \Rightarrow Q$ **conf** P

By using theorem 1

$$\begin{aligned} &Q \textbf{ conf } P \\ \Leftrightarrow &\forall s \in Tr(P).\ Ref(Q, s) \subseteq Ref(P, s) \\ \Leftrightarrow &\forall s \in Tr(P).\ Acc(Q, s) \subset\subset Acc(P, s) \\ \Rightarrow &\forall s \in Tr(\mathcal{A}(P)).\ \forall u' \in \mathcal{A}(Q).\ \exists t' \in \mathcal{A}(P). \\ &u'.acc \subset\subset t'.acc. \end{aligned} \quad (1)$$

We must also prove that $t(p)\mathcal{R}t(q)$:

Since $s \in Tr(\mathcal{A}(P))$, there exist t_1, u_1 such that $t \xrightarrow{s} t_1, u \xrightarrow{s} u_1$. Suppose that $t_1\mathcal{R}u_1$:

$$\begin{aligned} &\{\forall t'_1 \in \mathcal{A}(P).t_1 \xrightarrow{a} t'_1 \Rightarrow \\ &\exists u'_1 \in \mathcal{A}(Q).u_1 \xrightarrow{a} u'_1 \wedge\ u'_1.acc \subset\subset t'_1.acc\} \end{aligned}$$

Let us prove $t'_1\mathcal{R}u'_1$.

Let a be as $(t'_1 \xrightarrow{a} t''_1 \wedge Tr(P) \subseteq Tr(Q)) \Rightarrow s.a \in Tr(Q)$.

$\mathcal{A}(Q)$ is deterministic. So $u''_1 = D(u'_1, a)$ is such that $u'_1 \xrightarrow{a} u''_1$.

By using result (1), $u''_1.acc \subset\subset t''_1.acc$.

So $t'_1\mathcal{R}u'_1$.

Finally, we have Q **ext** $P \Leftrightarrow \mathcal{A}(P) \subsetneqq^{\Pi}_{\langle\Psi_1,\emptyset\rangle} \mathcal{A}(Q)$

□

The proof of the reduction relation **red** is similar except that the simulation is expressed in the opposite way. i.e $\mathcal{A}(P) \subsetneqq^{\Pi}_{\langle\emptyset,\Psi\rangle} \mathcal{A}(Q)$ means that $\mathcal{A}(Q)$ simulates $\mathcal{A}(P)$.

This theorem allows extension and reduction relations to be calculated like simulation relations on transformed graphs, although a direct implementation of their initial definitions, based on a trace set inclusion, would have been P-space complete.

4 Implementation and results

This part gives an overview of the Java prototype that has been developed to implement the reduction and extension relations. In order to illustrate obtained results, we present a case study modelling a phone and the different models that may be set up during the incremental modelling approach. The relations being demonstrated on LTS, computation is obviously performed on LTS. Nevertheless, for every step of modelling, we present the UML state machine associated with LTS for two reasons: using UML model is more widespread than LTS and this approach gives a quick outline of our future work consisting in applying results of LTS comparison on UML models.

4.1 Implementation of extension and reduction relations

The Java prototype we have developed follows the computation approach of theorem 3. Consequently, the main implemented classes are *LTS* and *AGraph* (cf. figure 2):

- *LTS* class implements a LTS as a set of states derived from the *State* class and transitions derived from the *Transition* class. States are identified by a name (*name* attribute) and transitions are labelled by actions (*label* attribute). Operations to set up a LTS are *searchState, addState and addTransition*. In order to implement theorem 3, the function *isStrongSimulatedBy*

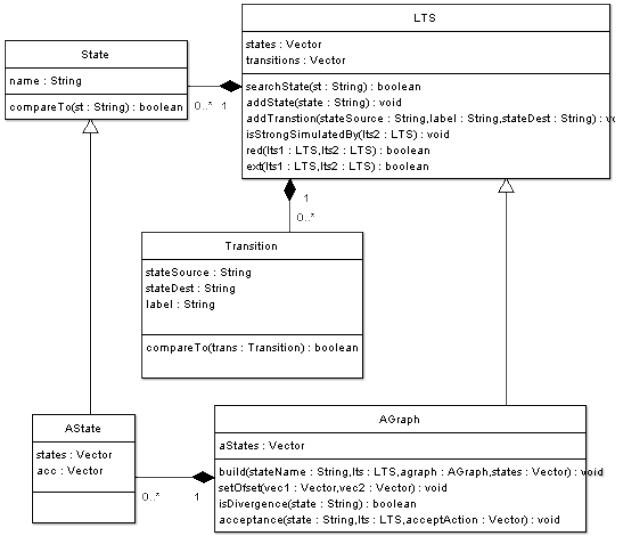

Figure 2. The class diagram of the prototype

have been implemented to check the simulation relation between two LTS.

- *AGraph* class implements the acceptance graph associated with a LTS. It is itself a LTS, the state of which belongs to the *AState* class defined as a subclass of *State* class. A *AState* state has the *states* attribute defining the list of its associated states in the LTS. This attribute allows the correspondences between acceptance graph nodes and LTS nodes to be established. Another fundamental attribute associated with a *AState* node is *acc* which is its acceptance set defined has a set of sets of actions.

Main steps of the prototype consist in building (function *build*) acceptance AGraphs associated with the two to be compared LTS. Then, the extension or reduction relation are computed (functions *ext* and *red*) by checking the simulation relation between the two AGraphs (function *isStrongSimulatedBy*) and the inclusion of acceptance sets (function setOfset) for every state of the AGraphs.

4.2 Case study: modelling a phone into three steps

This case study illustrates how to check the extension relation on a telephone model. Let us suppose that, at first, the modeller builds up a simple model outlining the main functionalities of a phone from a user point of view (see class Telephone, figure 3). Figure 4 represents the state machine SM_0 associated with this simple view. There are two functionalities:

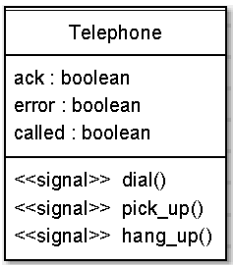

Figure 3. The class diagram of telephone

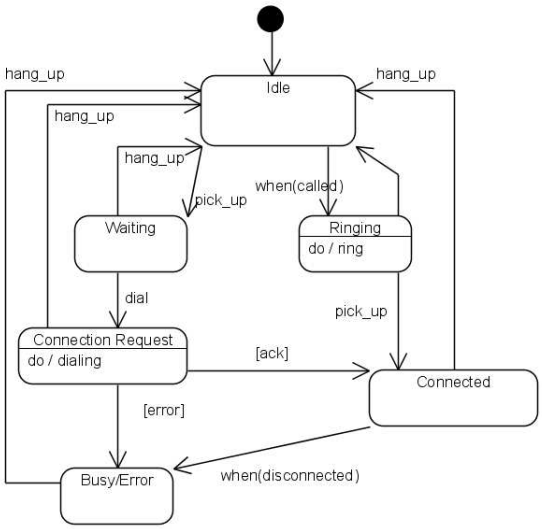

Figure 4. SM_0, the initial state machine of telephone

- The user is calling (left part of SM_0). In this case, the user picks-up and dials. The connection is thus requested and two cases may occur: the number is wrong or the called line is busy, and the called person picks-up. In the first case, the user can only hangs-up (transition *hang-up*). The second case leads to a connection (transition with a guard *ack*) that ends when the user hangs-up or when the calling person decides to stop the call.
- The user is called (right part of SM_0). If he picks-up, the connection is established and end as mentioned in the previous case. If he does not pick-up, the call ends when the calling person decides to stop the call.

Figure 5 shows the LTS, named LTS_0, associated with the state machine SM_0. Let us note that events that are not under the user control are labelled τ as internal transitions. Complete events, time events and change events in UML become LTS internal transitions. For instance,

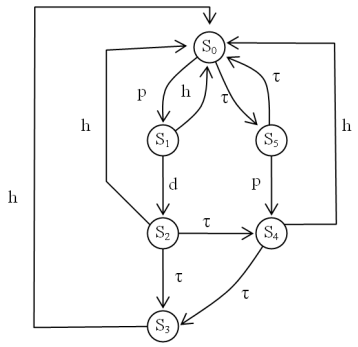

Figure 5. LTS_0

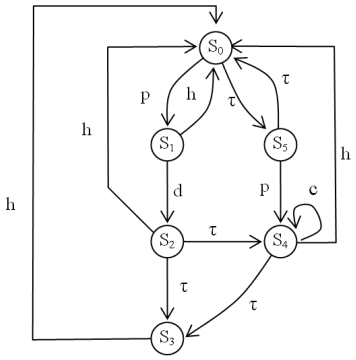

Figure 7. LTS_1

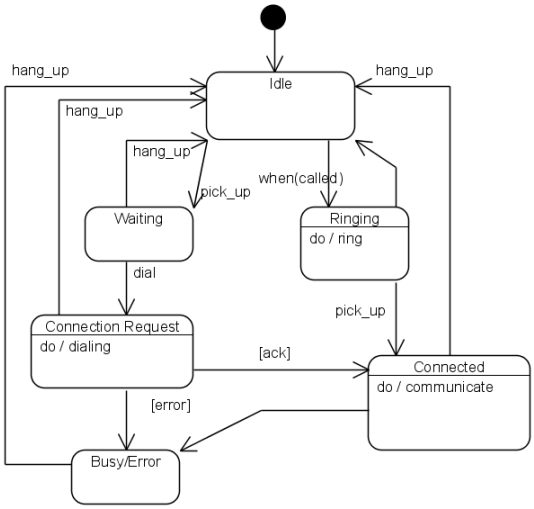

Figure 6. SM_1**, the state machine with communication activity**

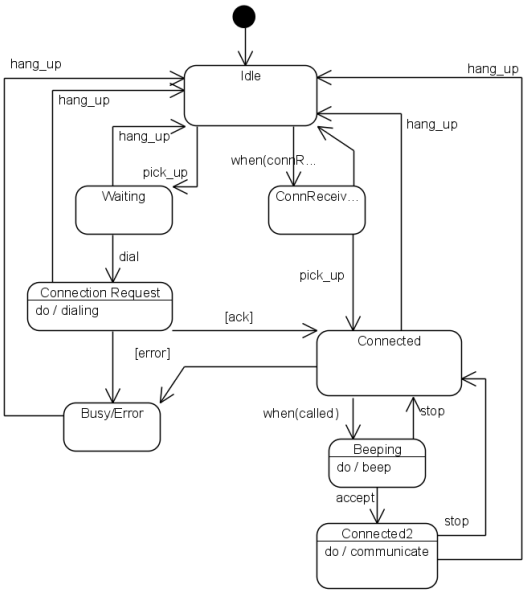

Figure 8. SM_2**, the state machine with second call**

the *when(called)* transition going out from the *Idle* state of SM_0 becomes a τ transition going out from state s_0 in LTS_0. Activities inside states may be observable or not.

Now, let us assume that the modeller wants to modify model SM_0 in order to highlight the communication activity of the phone, which is the most important. Model SM_0 is thus transformed into model SM_1 by adding an activity in state *Connected* named *communicate* (see figure 6). LTS_1 associated with SM_1 is the same as LTS_0 except the loop transition labelled *c* on state s_2 (cf. figure 7)

At last, the modeller wants to extend the functionalities of the phone and assume that a second call may be answered by the user while he is connected. Model SM_2 (figure 8) represents this new functionality by adding a *when(called)* transition to state *Connected*. Two cases may occur: the user does not accept the second call and stops it (transition *stop* of state *Beeping*) or the user accepts the second call and interrupts the first one (transition *accept* of state *Beeping*). In this last case, when the second call ends (transitions *stop* of state *Connected2*), the phone comes back in state *Connected*, except if the user hangs-up (transition *hang-up* of state *Connected2*). Figure 9 represents LTS_2 corresponding to SM_2.

4.3 Analysis of phone models by extension relation

In this part, we analyze if there is an extension relation from model SM_0 to SM_1, and from model SM_1 to SM_2. In the case where the model does not pass the checking, it

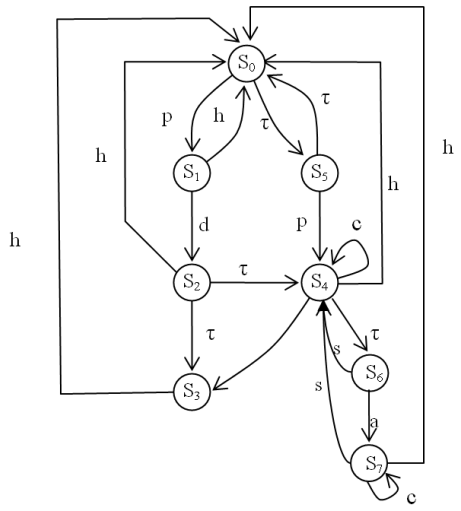

Figure 9. LTS_2

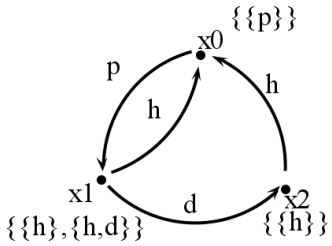

Figure 10. $AGraph_0$**, the acceptance graph of** LTS_0

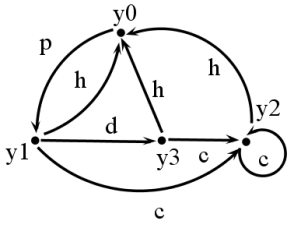

Figure 11. $AGraph_1$**, the acceptance graph of** LTS_1

has to be analyzed and modified.

4.3.1 Does SM_1 extend SM_0?

Let us consider LTS_0 and LTS_1 (figure 5 and 7). Figure 10 (resp. figure 11) represents $AGraph_0$ (resp. $AGraph_1$), the acceptance graph associated with LTS_0 (resp. LTS_1). In these graphs, transitions are labelled by actions and nodes are labelled by their name and their acceptance set, and their associated states are indicated, as in table 2.

AState	Associated states	Acceptance set
y_0	s_5, s_0	$\{\{p\}\}$
y_1	s_1, s_3, s_4	$\{\{h,d\},\{h\},\{h,c\}\}$
y_2	s_2, s_3, s_4	$\{\{h\},\{h,c\}\}$
y_3	s_3, s_4	$\{\{h\},\{h,c\}\}$

Table 2. Associated states and acceptance sets in $AGraph_1$

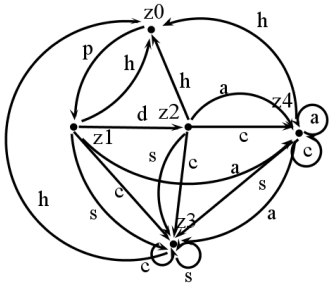

Figure 12. $AGraph_2$**, the acceptance graph of** LTS_2

The computed set of simulation relation is $\{(x_0, y_0), (x_1, y_1), (x_2, y_2)\}$, meaning that each node of LTS_0 is simulated by a node of LTS_1. The simulation relation is thus verified. Furthermore, the inclusion of acceptance sets is verified for each node. For example $Acc(y_2) = \{\{h\},\{h,c\}\} \subset\subset \{\{h\}\} = Acc(x_2)$. LTS_1 extends LTS_0 is computed by our tool.

4.3.2 Does SM_2 extend SM_1?

Let us consider LTS_1 and LTS_2 (figure 7 and 9). Figure 12 represents $AGraph_2$, associated with LTS_2, whose acceptance sets are represented in table 3.

AState	Associated states	Acceptance set
z_0	s_5, s_0	$\{\{p\}\}$
z_1	s_1, s_3, s_6, s_4	$\{\{h,d\},\{h\},\{s,a\},\{h,c\}\}$
z_2	s_2, s_3, s_6, s_4	$\{\{h\},\{s,a\},\{h,c\}\}$
z_3	s_3, s_6, s_4	$\{\{h\},\{s,a\},\{h,c\}\}$
z_4	s_3, s_6, s_4, s_7	$\{\{h\},\{s,a\},\{h,c\},\{c,s\}\}$

Table 3. Associated states and acceptance sets in $AGraph_2$

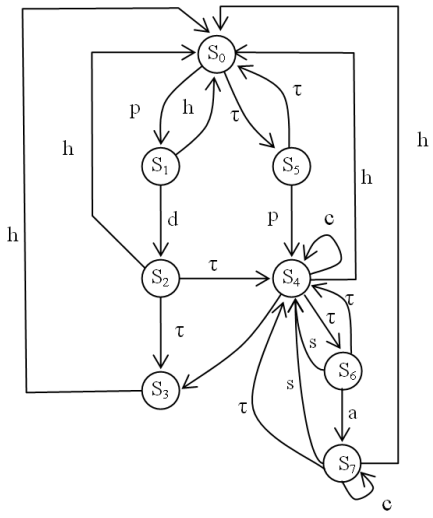

Figure 13. LTS_2^***, the corrected** LTS_2

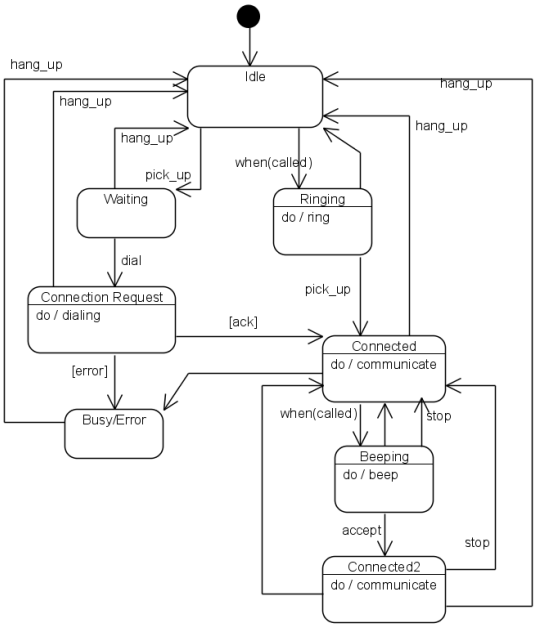

Figure 14. SM_2^***, the corrected** LTS_2

The computed set of simulation relation is $\{(y_0, z_0), (y_1, z_1), (y_2, z_2), (y_3, z_3)\}$ meaning that each node of LTS_1 is simulated by a node of LTS_2. The simulation relation is thus verified. However, the inclusion of acceptance sets is not verified for node z_3 (cf. table 3: $Acc(z_3) \subset\not\subset Acc(y_3)$). So LTS_2 does not extend LTS_1. The reason of failure has to be analyzed in order to improve model SM_2.

4.3.3 SM_2^*, the corrected SM_2

Analysis of $AGraph_2$ points out that acceptance set of node z_3 is $\{\{h\}, \{s, a\}, \{h, c\}\}$ while the one of node y_3 in $AGraph_1$ is $\{\{h\}, \{h, c\}\}$. The reason is that after the trace *pick-up;dial* (transitions *p* and *d* of LTS_2 in figure 9), the phone may refuse the action *hang-up* while in previous model LTS_1, this action cannot be refused. It means that the modeller made a mistake in model SM_2. He forgot that transition from state *Beeping* to *Connected* may be controlled by the user who made the second call (see model SM_2^* in Figure 14). The corresponding LTS_2^* is represented in figure 13: an internal transition from s_6 to s_4 has been added. The acceptance $AGraph_2^*$ graph of LTS_2^*, is the same as $AGraph_2$. Only acceptance sets are modified as shown in table 4. The new acceptance set associated with node s_6 is now $\{\{h\}, \{h, c, s, a\}\}$ and the property $\{\{h\}, \{h, c, s, a\}\} \subset\subset \{\{h\}, \{h, c\}\}$ is verified. As LTS_2^* simulates LTS_1, and the property of acceptance set inclusion is verified, SM_2^* extends SM_1.

AState	Associated states	Acceptance set
z_0	s_5, s_0	$\{\{p\}\}$
z_1	s_1, s_3, s_6, s_4	$\{\{h, d\}, \{h\}, \{h, c, s, a\}\}$
z_2	s_2, s_3, s_6, s_4	$\{\{h\}, \{s, a, h, c\}\}$
z_3	s_3, s_6, s_4	$\{\{h\}, \{h, c, s, a\}\}$
z_4	s_3, s_6, s_4, s_7	$\{\{h\}, \{s, h, c, a\}, \{c, s\}\}$

Table 4. Associated states and acceptance sets in $AGraph_2^*$

5 Conclusions and future works

Whereas the construction of programs following an incremental approach is a usual and encouraged means of development, the same kind of process for model development is rarely possible. In particular, UML behavioural models lack evaluation means, which are necessary to assess intermediate development steps.

Our goal was to propose evaluation techniques and tools for state machines, applicable during the development process. In the Labelled Transition System theory, comparison relations between systems have been defined. They can be used to compare a model of an implementation with a model of its specification, to check whether requirements are fulfilled or not. Hence, such relations can also be used to follow refinement approaches: a development step S_2 is said to refine a step S_1 if all implementations of S_2 are implementations of S_1. The extension relation defined over

LTS respects this property.

However, the complexity of the extension relation is such that no LTS tool implement it. In this paper, we propose a way to implement this extension relation, through a transformation into acceptance graphs. This is also suitable to implement the reduction relation.

We illustrate on a simple example how to use such an extension relation for an incremental development.

These works can lead us in a short future to propose an implementation of the central conformance relation. Future works concern also the adaptation of these relations and techniques to the full UML state machines, combined with UML sequence diagrams.

References

[1] OMG, *Unified Modeling Language Specification*, 2007.

[2] E. Boiten and M. Bujorianu, "Exploring UML refinement through unification," in *Critical Systems Development with UML - Proceedings of the UML'03 workshop*, J. Jürjens, B. Rumpe, R. France, and E. Fernandez, Eds., no. TUM-I0323. Technische Universität München, September 2003, pp. 47–62.

[3] M. V. D. Beeck, "Behaviour specifications: Equivalence and refinement notions," Techreport 24/00-I, Universität Münster, November 2000.

[4] C. Laneve and L. Padovani, "The must preorder revisited," *CONCUR 07*, 2007.

[5] R. Cleaveland and M. Hennessy, "Testing equivalence as a bisimulation equivalence," *Formal Aspects of Computing*, vol. 3, 1992.

[6] T. Lambolais and O. Gout, "Using conformance relations to help the development of state-machines," *ISSRE 04, International Symposium on Software Reliability Engineering*, Nov. 2004.

[7] O. Gout, "Développement incrémental de spécifications orientées objets," Ph.D. dissertation, École de Mines d'Alès, 2006.

[8] R. Milner, *Communicating and Mobile Systems: The π Calculus*. Cambridge University Press, 1999.

[9] E. Brinksma and G. Scollo, "Formal Notions of Implementation and Conformance in LOTOS," Technical Report INF-86-13, Dept. of Informatics, Twente University of Technology, 1986.

[10] M. Hennessy, *Algebraic theory of processes*. Mit Press Series In The Foundations Of Computing, 1988, no. ISBN:0-262-08171-7.

[11] G. Leduc, "Conformance relation, associated equivalence, and minimum canonical tester in lotos," *PSTV XI. North-Holland*, 1991.

[12] J. Tretmans, "Testing concurrent systems: A formal approach," *CONCUR 99, LNCS*, no. 1664, 1999.