Semantics and implementation of a refinement relation for behavioural models

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

Semantics and implementation of a refinement relation for 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. In this paper, we present a way to implement refinement relations over transition systems, useful for incremental behavioural model development. Our work is based on the extension relation defined over Labelled Transition Systems (LTS). This relation appears to be suitable for refinement developments, but its computability has not yet been established. We propose an implementation which relies on the generalisation of a bisimulation relation applied on acceptance graphs. It is formally demonstrated and illustrated through a case study of a system modelled both in UML state machine and LTS. The analysis is performed on LTS by a JAVA prototype we have developed, and the interpretation of results is given on the corresponding UML state machine. Moreover, when the relation is not satisfied, the tool can exhibit failing traces and can give several proposals to improve the model. Hence, this work goes toward a behavioural semantics for the class specialisation relation of object oriented models.

1 Introduction

We are interested in the refinement of UML State Machines (UML SM). So far, the refinement notion has been well stated on formal specifications such as set theory based languages Z and B [1]. In the object orientation world, this is not a primary notion, even though the specialisation relation between classes has some similarities with a refinement relation. In behavioural models like automata's theory (Labelled Transition Systems—LTS, Input/Output Automata, Petri Nets, Process Algebras, ...), refinement is neither a built-in notion. In the UML community, some works address the problem of inter-consistency and intra-consistency [2]. Nevertheless, the refinement problem of State Machines is rarely addressed.

If we consider the simple language of LTS, it appears that an already defined relation is adequate for refinement purposes, in the sense that any implementation of a 'detailed' model is an implementation of the initial model. However, no efficient verification implementation of this relation has been proposed yet. In this paper, we propose an implementation technique which relies on bisimulation computations over transformed graphs, thanks to related works developed by [3].

The complementary side of these works consists in adapting these results on UML SM. Rather than defining a new refinement relation on UML SM, we choose to propose a transformation between UML SM and LTS, compare the obtained LTS with respect

to the chosen relation, and make an abstract interpretation of the comparison on UML models. This allows us to point out some modelling refinement mistakes, since a non checked relation on LTS means there is no refinement on UML SM.

The paper is organized in three parts. Section 2 presents existing relations over LTS and proposes an adequate one for refinement purposes, along to two criteria. This part is completed by formal definitions over LTS. Section 3 focuses on the refinement relation computability. For this purpose, it proposes a theorem which relates refinement to bisimulation computed over acceptance graphs. This leads to an efficient algorithm. In section 4, we present the demonstrator we have developed in JAVA. The obtained results are illustrated on a case study. An example of a non trivial mistake during refinement is pointed out thanks to the proposed relation verification. This part also presents the transformation principles from UML state machines to LTS. Section 5 concludes and presents our future works.

2 Analysis of existing relations

This section gives an informal presentation of preorders defined overs LTS and shows that the extension relation is a good candidate for a strong refinement relation. Then, we recall definitions of LTS [4], refusal sets [5], acceptance sets [6] and some associated results useful to give an implementation of refinement verification.

2.1 Toward a refinement relation to compare behavioural models

This part proposes a summary of existing relations developed for comparing behavioural models in order to point out which of them can be used in a framework of refinement modelling.

We are looking for asymmetric (but not necessarily antisymmetric) and transitive relations. The refinement technique we want to follow consists in adding concrete details and reducing non-determinism. A refined model is more precise (details are added) and more concrete (abstraction is reduced) step towards an implementation model.

Refinement relations for LTS. Here, we present informal interpretations of relations which will be formally defined in the next section. For the moment, we consider that LTS describe machines interacting with their environment by means of actions. A special (unobservable) action is also proposed to describe internal treatments.

Trace inclusion. In the case of LTS, adding details consists in defining new traces and possibly new actions. A trace is an observable and partial sequence of actions starting from the initial state. The problem that arises is that a LTS R can have more traces than a LTS A and be less deterministic (for instance, using Milner notations [4], $R =_{\text{def}} a; b; stop + a; stop$ has the same trace set as $A =_{\text{def}} a; b; stop$, but may fail after a).

Simulation relations. By the same way, simulation relations proposed by Milner [4] are not adequate for refinement and implementation purposes. Indeed, like trace inclusion, if R simulates A, R may only do what A must do, but may also refuse it. Again, R can be less deterministic than A. If we see A like a reference specification, we want to define a refinement R that should accept what A must accept.

Conformance relations. The relation $\mathbf{conf}$ has been proposed as an implementation relation to check if a model of an implementation fulfils its specification [7,8]. It is a formalisation of the conformance relation defined in conformance testing. An implementation model I conforms to a specification model S if it must do all what S must do (or if S may refuse all what I may refuse). The $\mathbf{conf}$ relation specifically captures the 'reduction of non determinism' but it is not transitive.

Extension and reduction relations [7,8] are conformance relations combined with trace inclusion. They respectively consist in extending and reducing the set of traces while preserving conformance. Both are transitive. The extension consists in adding new functionalities by preserving existing ones. The reduction consists in removing useless functionalities or adding constraints to existing ones, always by preserving core functionalities. This latter approach may seem more intricate, and is more rarely adopted in model development. The extension relation combines reduction of non-determinism and extension of traces, which is exactly what we are looking for.

Unfortunately these relations ($\mathbf{conf}$, $\mathbf{red}$ and $\mathbf{ext}$) appear to be hard to check in practice [9]. Interesting implementation results can be found for may and must preorders defined by Hennessy and Cleaveland [3]. May and must relations are similar to trace inclusion and reduction relation, but are defined to take also into account the cases of divergent automata (i.e. automate which could enter in infinite internal transition sequences). Cleaveland [3] proposed polynomial algorithms to implement may and must preorders.

The following table gives a summary of these relations according to two criteria: reduction of non determinism and extension of the set of traces.

	Reduction of non determinism	Extension of traces
$\mathbf{conf}$	$\star\star\star$	$\star$
$\mathbf{red}$	$\star\star\star$	$\emptyset$
$\mathbf{ext}$	$\star\star\star$	$\star\star\star$
$\mathbf{may}$	$\emptyset$	$\star\star\star$
$\mathbf{must}$	$\star\star\star$	$\emptyset$

$\star$ may be supported.
$\star\star\star$ is guaranteed.
$\emptyset$ is not supported.

The extension relation is the only one which matches the two targeted criteria. Let us examine in more details if it is suitable for refinement purposes. As stated by Guy Leduc [7], $\mathbf{conf}$ and $\mathbf{ext}$ are such that:

$$R\ \mathbf{ext}\ A \Rightarrow \forall P.\ (P\ \mathbf{conf}\ R \Rightarrow P\ \mathbf{conf}\ A). \tag{1}$$

Considering that $\mathbf{conf}$ is an implementation relation, this latter relation states that $\mathbf{ext}$ is stronger than a refinement relation $\sqsupseteq$, characterized by

$$R\ \sqsupseteq A \Leftrightarrow \forall P(P \operatorname{imp} R \Rightarrow P \operatorname{imp} A). \tag{2}$$

Where $R \sqsupseteq A$ means that R refines A. This property can also be formulated by [10]:

$$R \sqsupseteq A \Leftrightarrow \mathcal{M}(\mathcal{R}) \subseteq \mathcal{M}(\mathcal{A}), \tag{3}$$

where $\mathcal{M}(A)$ denotes all the implementations of A. This matches the B or Z definition of refinement [1].

The $\mathbf{ext}$ relation is a refinement relation, but it is not the largest one. In a context of model development, such a relation is useful in case it is satisfied (since it guarantees, then, that the refinement is a correct one), and can be used like a warning in case it is not satisfied.

Refinement relations for UML State Machines. We consider that UML State Machines are associated to classes. They are used to describe the expected behaviours of the objects of a class. On object oriented models, if a class C_R is a specialisation of a class C_A, which is written C_R $\mathbf{extends}$ C_A in some object oriented languages like JAVA, this means that all instances of C_R can behave like instances of C_A: any instance of C_R is also an instance of C_A. Instances of C_R can do everything that instances of C_A can also do. Stated more formally, we could write:

$$C_R \text{ extends } C_A \Rightarrow \text{Instances}(C_R) \subseteq \text{Instances}(C_A). \tag{4}$$

More precise definitions could be found in [11]. Properties (1) and (4) are similar. This means that the extension relation over classes is a refinement relation. But what about class refinement and their associated state machines refinement? What kind of relation can we define to state that a state machine SM_R refines a state machine SM_A, and how could we check this relation? Since UML state machines can be seen like extensions of simple labelled transition systems, our point of view is to consider LTS like an abstract interpretation of state machines. We won't define a detailed relation over State Machines (like we did for conformance [12]). Rather, we shall elaborate a translation from UML to LTS and check whether extension is satisfied or not between LTSs. The backward interpretation leads to point out some abstract results (the one carried by LTS models).

2.2 Formal definitions of conformance relations

A LTS [4] is a graph consisting of states linked by labelled transitions. It models behavioural specifications as well as implementations.

Definition 1 (Labelled Transition Systems). *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. Let P and Q be two LTS. We need the following notations:

$$s \xrightarrow{a} s' =_{def} (s, a, s') \in \rightarrow$$
$$s \xrightarrow{a_1 \cdots a_2} s' =_{def} \exists s_0, \ldots, s_n.\ s = s_0 \xrightarrow{a_1} \ldots \xrightarrow{a_n} s_n = s'$$

$$
\begin{aligned}
s \xrightarrow{a_1 \cdots a_2} &=_{def} \exists s'.\ s \xrightarrow{a_1 \cdots a_n} s' \\
s \stackrel{\epsilon}{\Longrightarrow} s' &=_{def} s = s' \text{or } s \xrightarrow{\tau \cdots \tau} s' \\
s \stackrel{a}{\Longrightarrow} s' &=_{def} \exists s_1, s_2.\ s \stackrel{\epsilon}{\Longrightarrow} s_1 \xrightarrow{a} s_2 \stackrel{\epsilon}{\Longrightarrow} s' \\
s \stackrel{a_1 \cdots a_2}{\Longrightarrow} s' &=_{def} \exists s_0, \ldots, s_n.\ s = s_0 \stackrel{a_1}{\Longrightarrow} \ldots \stackrel{a_n}{\Longrightarrow} s_n = s' \\
s \stackrel{\sigma}{\Longrightarrow} &=_{def} \exists s'.\ s \stackrel{\sigma}{\Longrightarrow} s' \\
s \text{ after } \sigma &=_{def} \{s' \mid s \stackrel{\sigma}{\Longrightarrow} s'\} \\
P \text{ after } \sigma &=_{def} s_0 \text{ after } \sigma \\
\text{Traces : } Tr(P) &=_{def} \{\sigma \in L^* \mid s_0 \stackrel{\sigma}{\Longrightarrow}\} \\
Out(p) &=_{def} \{a \in L \mid p \xrightarrow{a}\} \\
Out(p, \sigma) &=_{def} \cup_{p' \in p \text{ after } \sigma} Out(p') \\
Out(P, \sigma) &=_{def} Out(s_0, \sigma) \\
D(s, a) &=_{def} \{s' \mid s \xrightarrow{a} s'\}
\end{aligned}
$$

The refusal set of P after trace σ is defined by:

Definition 2 (Refusal set). $Ref(P, \sigma) =_{def} \left\{X \mid \exists p \in P \textit{ after } \sigma.\ p \stackrel{\sigma}{\not\Longrightarrow}, \forall e \in X\right\}$.

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

Definition 3 (Conformance). Q **conf** P *if* $\forall \sigma \in Tr(P).\ Ref(Q, \sigma) \subseteq Ref(P, \sigma)$.

Extension and reduction are defined as extending or reducing traces, while preserving the conformance.

Definition 4 (Reduction). Q **red** P *if* $Tr(Q) \subseteq Tr(P)$ *and* Q **conf** P.

Definition 5 (Extension). 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.

2.3 Acceptance sets

In this section, we present a definition of acceptance sets and their relation with refusal sets. This notion will be used in the next section to build up acceptance graphs associated to LTS.

Definition 6 (Acceptance set). *The acceptance set of P after σ is defined by:*

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

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 of the acceptance set 6:

$$\begin{aligned}
&\forall \sigma \in Tr(P).\ Acc(P,\sigma) \subset\subset Acc(Q,\sigma) \\
\Leftrightarrow &\forall \sigma \in Tr(P).\ \forall X \in Acc(P,\sigma).\ \exists Y \in Acc(Q,\sigma).\ Y \subseteq X \\
\Leftrightarrow &\forall \sigma \in Tr(P).\ \forall X \in \{Out(p',\epsilon) | P \stackrel{\sigma}{\Longrightarrow} p'\}.\ \exists Y \in \{Out(q',\epsilon) | Q \stackrel{\sigma}{\Longrightarrow} q'\}.\ Y \subseteq X \quad (7)
\end{aligned}$$

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:

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

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

3 Computability of extension and reduction relations

After having selected the most appropriate relations to compare models and introduced main definitions of the domain, we outline the problem of computability of the relations $\mathbf{ext}$ and $\mathbf{red}$. As far as we know, these relations have not been implemented yet. Nevertheless, Cleaveland and Hennessy [3] 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

Inspired of the Cleaveland's work of generalisation of prebisimulation definition [3], we reformulate the definition of bisimulation given by Milner [4].

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* $p, q \in S$, $p\mathcal{R}q$ *implies:*

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.).
$P \subsetapprox^{\Pi}_{\langle \Psi_1, \Psi_2\rangle} Q$ *if there 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 relation Π defined over two states.

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:*

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

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* $t_1 \xrightarrow{a}_T t_2$;
5. $t_0 = (\{p_0\})^{\epsilon}$.

This definition is similar to acceptance graphs of [3], but the definition of acceptance sets does not take into account divergence states. The algorithm of acceptance graphs construction introduced by [3] can be adapted to the construction of acceptance graphs as defined in this paper.

3.3 Demonstration of the extension and reduction computability

Theorem 2. *Let P be a LTS and $\mathcal{A}(P)$ be 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** $P \Leftrightarrow \mathcal{A}(P) \subsetapprox^{\Pi}_{\langle \emptyset, \Psi_2\rangle} \mathcal{A}(Q)$
2. Q **ext** $P \Leftrightarrow \mathcal{A}(P) \subsetapprox^{\Pi}_{\langle \Psi_1, \emptyset\rangle} \mathcal{A}(Q)$

Proof. We are going to prove (2.). (1.) is similar and will not be expressed in this article. We note t is a state $\in \mathcal{A}(P)$ and $u \in \mathcal{A}(Q)$

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

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

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

We have to demonstrate the trace inclusion which is in the definition of $\mathbf{ext}$. Hence, the first part of the relation $\mathcal{R}$ can be rewritten:

$$\begin{aligned}
&\forall t' \in \mathcal{A}(P).t \xrightarrow{a}_T t' \Rightarrow \exists u' \in \mathcal{A}(Q).u \xrightarrow{a}_T u' \\
\Rightarrow &\forall a \in \sigma \wedge \sigma \in Tr(\mathcal{A}(P)) \Rightarrow \exists \sigma \in Tr(\mathcal{A}(Q)) \\
\Rightarrow &Tr(\mathcal{A}(P)) \subseteq Tr(\mathcal{A}(Q)) \Rightarrow Tr(P) \subseteq Tr(Q).\text{(Theorem 2)}
\end{aligned}$$

To continue to demonstrate the conformance, we use the definition of acceptance graph $u'.acc = Acc(Q,\sigma), t'.acc = Acc(P,\sigma)$ and theorem 1. So the second part of the relation $\mathcal{R}$ can be reformulated:

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

We have therefore the conformance: $\mathcal{A}(P) \lessapprox^{\Pi}_{\langle \Psi_1, \emptyset\rangle} \mathcal{A}(Q) \Rightarrow Q\ \mathbf{ext}\ P$

$\Rightarrow$)

From $Q\ \mathbf{ext}\ P$, we must prove the relation $\mathcal{R}\langle \Pi, \Psi_1, \emptyset\rangle$ as defined above.

By using theorem 1, we try to give the formula of the strong simulation relation.

$$\begin{aligned}
&Q\ \mathbf{ext}\ P \Rightarrow Q\ \mathbf{conf}\ P \\
\Leftrightarrow &\forall \sigma \in Tr(P).\ Ref(Q,\sigma) \subseteq Ref(P,\sigma) \\
\Leftrightarrow &\forall \sigma \in Tr(P).\ Acc(Q,\sigma) \subset\subset Acc(P,\sigma)(\text{Theorem 1}) \\
\Rightarrow &\{\forall \sigma \in Tr(\mathcal{A}(P)).t_0 \xrightarrow{\sigma}_T t_n \Rightarrow (\exists u_n.u \xrightarrow{\sigma}_T u_n) \wedge\ (u_n.acc \subset\subset t_n.acc)\}. \quad (5)
\end{aligned}$$

To prove the relation $\mathcal{R}$ as established above, we must prove $t_n\ \mathcal{R}\ u_n$.

$$\begin{aligned}
&\text{From (5): } \{\forall \sigma \in Tr(\mathcal{A}(P)).\ t_0 \xrightarrow{\sigma}_T t_n.\ \Rightarrow \exists u_n.u_0 \xrightarrow{\sigma}_T u_n\} \\
&\text{Suppose that } a \in t_n.acc \Rightarrow a \in u_n.acc \text{ and } \sigma.a \in Tr(P) \subseteq Tr(Q) \\
\Rightarrow &\{t_n \xrightarrow{a} t'_n \Rightarrow \exists u'_n.\ u_n \xrightarrow{a} u'_n \wedge\ u'_n.acc \subset\subset t'_n.acc\}
\end{aligned}$$

Let us prove $t'_n\ \mathcal{R}\ u'_n$. Suppose that $t'_n \xrightarrow{b}_T t''_n$.

$$\sigma.a \in Tr(P) \subseteq Tr(Q) \wedge b \in u'_n.acc \subset\subset t'_n.acc \Rightarrow \sigma.a.b \in Tr(P) \subseteq Tr(Q).$$

$$\begin{aligned}
&\text{Because } \mathcal{A}(Q) \text{ is deterministic } \Rightarrow \exists u''_n.\ u'_n \xrightarrow{b}_T u''_n. \\
\Rightarrow &u''_n.acc \subset\subset t''_n.acc \text{ (By using (5))} \\
\Rightarrow &t'_n \xrightarrow{b}_T t''_n.\exists u''_n.\ u'_n \xrightarrow{b}_T u''_n.u''_n.acc \subset\subset t''_n.acc \\
\Rightarrow &t'_n\ \mathcal{R}\ u'_n
\end{aligned}$$

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

The extension relation can be simply interpreted such as the simulation between two transformed graphs ($\mathcal{A}(Q)$ simulates $\mathcal{A}(P)$), and at each simulation states pair, there exists the inclusion of acceptance set ($\Pi = \{\langle t, u\rangle | u.acc \subset\subset t.acc\}$).

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

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 we have developed to implement the 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.

4.1 Implementation of extension and reduction relations

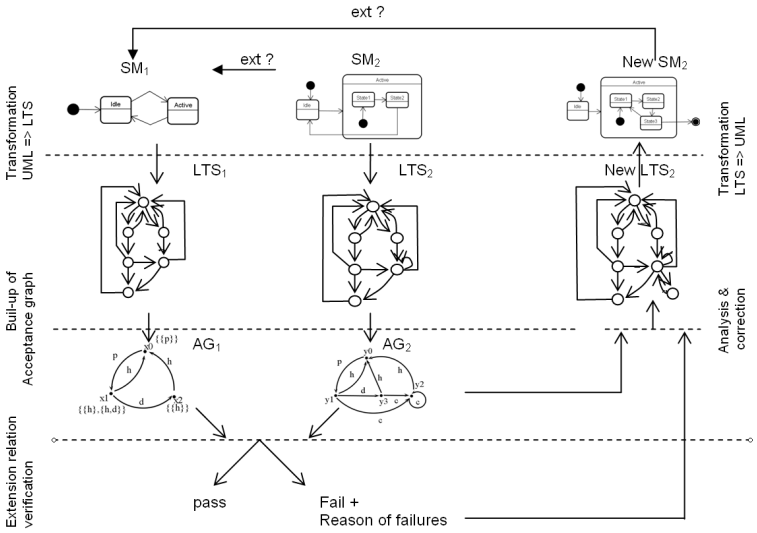


Fig. 1. Overview of the computation of extension relation

The JAVA prototype we have developed follows the computation approach of theorem 3. Consequently, the main steps to compute the extension relation are (see Figure 1):

– Transformation of state machines to be analysed into their corresponding LTS. This step is manually achieved using informal rules presented in [13] and summarised in Figure 2. Note that, at present, the UML state machines we consider do not take into account signal or method parameters, neither other variables. Hence, UML guards are not taken into account in corresponding LTS, and systematically lead to non deterministic transitions.
– Computation of the acceptance graph associated to LTSs to be compared. This step is automatic and leads to associate an acceptance set to every node of the graph.
– Computation of the bisimulation relation for every state of the acceptance graph and verification of the acceptance sets inclusion.

UML	visible action	hidden action
time or change event [guard]/action	$\xrightarrow{\tau} . \xrightarrow{a}$	$\xrightarrow{\tau}$
call or signal event [guard]/action	$\xrightarrow{e} \cdot \xrightarrow{a}$	$\xrightarrow{e}$
[guard]/action	$\xrightarrow{\tau} \cdot \xrightarrow{a}$	$\xrightarrow{\tau}$
/action	$\xrightarrow{\tau} \cdot \xrightarrow{a}$	$\xrightarrow{\tau}$

Fig. 2. Rules to transform transitions of UML state machines into LTS

The last step leads to guarantee that the model under development is an extension of the reference model or not. If it is not the case, the verification tool gives details about the state which failed and the reason why. By this way, the designer is guided to modify the LTS of the model under development. Consequently, he is able to enhance the UML model. A new verification loop is entering in order to test again the new model, until the extension relation is verified. Next section illustrates this iterative modelling approach.

4.2 Case study: modelling simple and double call phones

This case study concerns the incremental modelling process of a phone and the verification of relations existing between classes developed at each step. A UML class diagram presenting phone classes and interfaces is shown in Figure 3. The first step consists in defining the specification of a simple phone from the user point of view (see class $SimplePhone$ in Figure 3). The specification required interface are user actions: *hang_up*, *pick_up*, *dial* and *comm_in* (incoming communication). These are signals the $SimplePhone$ is subscribed to. Signals provided by the phone and intended to the user are shown in interface *User_msg*. Let us suppose that class $SimplePhone$ is considered as a reference model. Next modelling step consists in extending the functionality of $SimplePhone$ in order to specify a phone accepting a double call while the user is on the phone. The class $DoubleCall$ (Figure 3) is a specialisation of the $SimplePhone$ class. It inherits $SimplePhone$ interfaces, and has its own required interface defining signals accepting or rejecting/stopping the second call. In order to follow results of reduction analysis between $SimplePhone$ and $DoubleCall$, let us consider state machines associated with these classes (Figure 4).

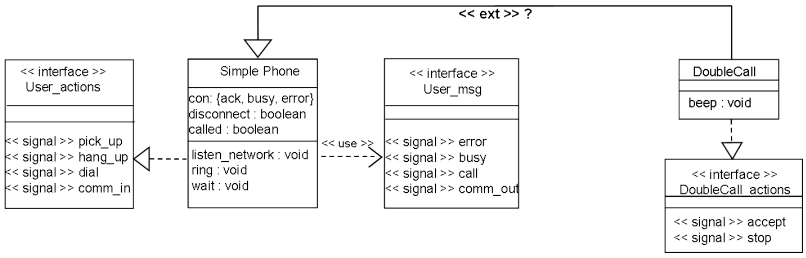


Fig. 3. Class diagram of simple and double call phones

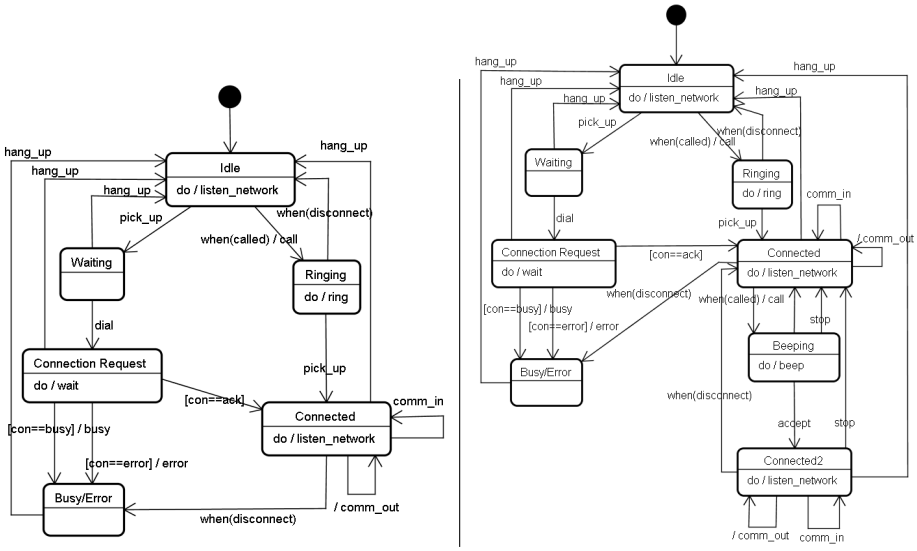


Fig. 4. (a) $SM_{SimplePhone}$ (b) $SM_{DoubleCall}$

$SM_{SimplePhone}$ (Figure 4.a) represents the state machine associated with $SimplePhone$ class. There are two functionalities:

- The user is calling (left part of $SM_{SimplePhone}$). 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 $SM_{SimplePhone}$). 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.

$SM_{DoubleCall}$ (Figure 4.b) is the same as $SM_{SimplePhone}$ apart from state *Connected*: there is a new transition named *when(called)* in order to model the second call.

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*).

The goal is to verify if there is an extension relation between the simple call phone and the double call one.

4.3 Does $SM_{DoubleCall}$ extend $SM_{SimplePhone}$?

The result given by our tool is that $LTS_{DoubleCall}$ does not extend $LTS_{SimplePhone}$.

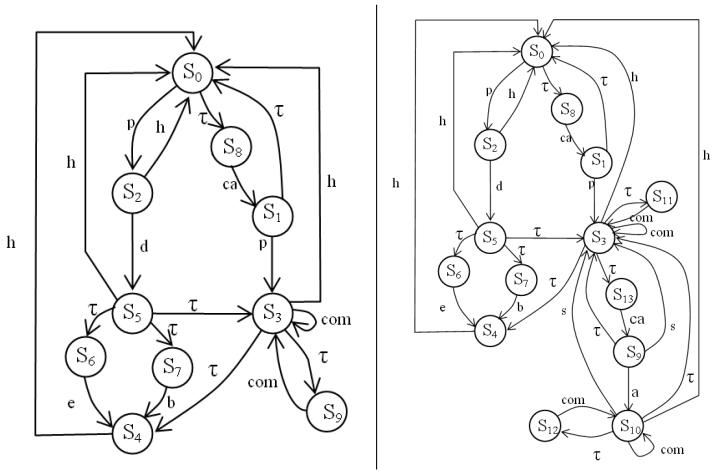
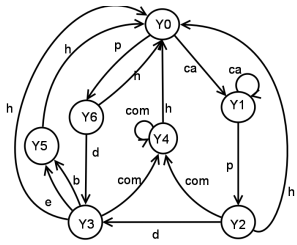


Fig. 5. (a) $LTS_{SimplePhone}$ (b) $LTS_{DoubleCall}$

We give in details intermediate computations in order to illustrate the application of theorem 3 as performed by the tool (see Figure 1). Figure 5 shows the LTS associated with state machines $SM_{SimplePhone}$ and $SM_{DoubleCall}$, respectively named $LTS_{SimplePhone}$ and $LTS_{DoubleCall}$. LTS actions are named by the first letters of UML labels. UML signals which are required *and* provided (such as *comm_in* and *comm_out*) are translated into a single LTS action (*com*).

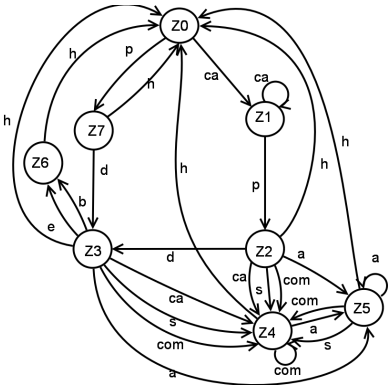
Having defined LTS associated with state machines to be compared, the analysis tool is run in order to build up acceptance graphs and compute the bisimulation relation. Figure 6 (resp. Figure 7) represents the acceptance graph associated with $LTS_{SimplePhone}$ (resp. $LTS_{DoubleCall}$). In these graphs, transitions are labelled by actions defined in the LTS. Tables of Figures 6 and 7 give the acceptance set associated to every node of the acceptance graphs.

Two properties are automatically checked. The first one is that there exists, for each node of $\mathcal{A}(SimplePhone)$, a node of $\mathcal{A}(DoubleCall)$ which simulates it. The result of



AState	Acceptance set
x_0	$\{\{ca\},\{ca,p\}\}$
x_1	$\{\{ca\},\{ca,p\}\}$
x_2	$\{\{d,h\},\{com\},\{com,h\},\{h\}\}$
x_3	$\{\{e\},\{com\},\{b\},\{com,e,b,h\},$ $\{com,h\},\{h\}\}$
x_4	$\{\{com\},\{com,h\},\{h\}\}$
x_5	$\{\{h\}\}$
x_6	$\{\{d,h\}\}$

Fig. 6. $\mathcal{A}(SimplePhone)$ and its acceptance sets



AState	Acceptance set
z_0	$\{\{ca\},\{ca,p\}\}$
z_1	$\{\{ca\},\{ca,p\}\}$
z_2	$\{\{d,h\},\{com\},\{ca\},\{h\},$ $\{ca,com,,s,a,h\}\}$
z_3	$\{\{com\},\{e\},\{ca\},\{ca,com,s,a,h\},$ $\{b\},\{e,ca,com,b,s,a,h\},\{h\}\}$
z_4	$\{\{com\},\{ca\},\{ca,com,s,a,h\},\{h\}\}$
z_5	$\{\{ca\},\{com\},\{ca,com,s,a,h\},\{h\}\}$
z_6	$\{\{h\}\}$
z_7	$\{\{d,h\}\}$

Fig. 7. $\mathcal{A}(DoubleCall)$ and its acceptance set

this first property is expressed by a set of pairs representing simulation relationships. In this case, it is: $\{(x_4,z_5),(x_5,z_8),(x_6,z_7),(x_3,z_3),(x_2,z_2),(x_1,z_1),(x_7,z_9),(x_0,z_0)\}$. The second property is that the acceptance set of each node of $\mathcal{A}(DoubleCall)$ has to be included into the acceptance set of its associated node in $\mathcal{A}(SimplePhone)$. This property is checked by analysing acceptance sets given in tables of Figures 6 and 7 for each pair belonging to the simulation relation. However, the inclusion is not verified for the first simulation pair (x_4,z_5) since $Acc(z_5) \subset\not\subset Acc(y_4)$. So $LTS_{DoubleCall}$ does not extend $LTS_{SimplePhone}$. The reason of failure has to be analysed in order to improve model $SM_{DoubleCall}$.

4.4 Improving double call specification

Let us examine in details the reason of the failure highlighted by the verification tool between $LTS_{SimplePhone}$ and $LTS_{DoubleCall}$: $Acc(z_5) \subset\not\subset Acc(x_4)$. By analysing acceptance sets of nodes x_4 and z_5 (see tables in Figures 6 and 7), we can observe that after traces $p;d$ or p, $DoubleCall$ machine may refuse h and com while $SimplePhone$ may refuse h or com, but not both. Solutions can thus be automatically proposed to the designer to fulfil the inclusion property: actions com, h or both have to be added to node z_5. Following the Occam's Razor, the designer is advised to add only one action

(com or h) outgoing from state s_{13} which is the state associated to z_5 whom acceptance set is limited to $\{ca\}$. The mistake is corrected on the LTS. Then, the issue is to find the corresponding element on the UML state machine. In this case, s_{13} has been introduced as an intermediate state for modelling the state occurring after the event $when(called)$. The corresponding correction on the UML state machine consists in decomposing the $when(called)/call$ transition into two transitions and adding a new state that may accept action $call$ and action com or h. Let us suppose that action com is selected. Figure 8 shows the modified part of $SM_{DoubleCall^*}$, the new UML model obtained after the correction.

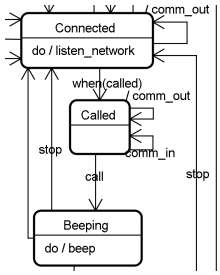


Fig. 8. Modified part of $SM_{DoubleCall^*}$, the corrected version of $SM_{DoubleCall}$

The analysis is again performed between the new acceptance graph of $DoublePhone^*$ and the acceptance graph of $SimplePhone$.

In this case, the simulation relation is the same as previously and acceptance sets associated to simulated states fulfil the inclusion property. Thus, $LTS_{DoublePhone^*}$ is guaranteed to extend $LTS_{SimplePhone}$.

4.5 Conclusion about the phone case study

We have defined a class $DoubleCall^*$ representing a high-level specification of a double call phone. It has been demonstrated that this specification is an extension of the simple call phone. Future modelling step could consist in defining an implementation of this class and verifying its conformance using the tool we have developed [13]. Since the extension relation is a refinement relation (see property (1) in section 2.1), if an implementation of $DoubleCall^*$ conforms to its specification, then it surely conforms to $SimplePhone$. It means that any $DoubleCall^*$ implementation is also an implementation of $SimplePhone$.

5 Conclusions and future works

In this paper, we focused on the extension relation over LTS which is a refinement relation, and proposed an efficient way to implement its verification. We have also proposed translation schemes from UML SM to LTS. We have illustrated results obtained

by our JAVA prototype on a simple but representative example. Since UML can describe more detailed behavioural models, abstract results obtained on LTS verification have to be considered like warnings on UML SM. As illustrated by the case study, when the relation is not satisfied, exhibited failing traces help us find correction ways.

In previous works [13], we have proposed verification techniques for the implementation relation. Combined with current works, this helps define a semantics to UML SM specialisation and implementation.

Future work will address two issues. The first one concerns the refinement relation itself. The extension relation over LTS is not the largest refinement one. We are looking for an implementation of the exact relation '*refines*', defined as follows:

$$R \textit{ refines } A \Leftrightarrow \forall P.\ (P \textbf{ conf } R \Rightarrow P \textbf{ conf } A). \tag{6}$$

The second issue concerns a more formal approach to translate UML SM into LTS. This work can benefit from the MDE approach.

References

1. Abrial, J.R.: The B-Book: Assigning Programs to Meanings. Cambridge University Press (1996)
2. Huzar, Z., Kuzniarz, L., Reggio, G., Sourrouille, J.: Consistency Problems in UML-Based Software Development. UML 2004 Satellite Activities (2005)
3. Cleaveland, R., Hennessy, M.: Testing equivalence as a bisimulation equivalence. Formal Aspects of Computing **3** (1992)
4. Milner, R.: Communicating and Mobile Systems: The π Calculus. Cambridge University Press (1999)
5. Brinksma, E., Scollo, G.: Formal Notions of Implementation and Conformance in LOTOS. Technical Report INF-86-13, Dept. of Informatics, Twente University of Technology (1986)
6. Hennessy, M.: Algebraic theory of processes. Number ISBN:0-262-08171-7. Mit Press Series In The Foundations Of Computing (1988)
7. Leduc, G.: Conformance relation, associated equivalence, and minimum canonical tester in lotos. PSTV XI. North-Holland (1991)
8. Tretmans, J.: Testing concurrent systems: A formal approach. CONCUR 99, LNCS (1664) (1999)
9. Laroussinie, F., Schnoebelen, P.: The State Explosion Problem from Trace to Bisimulation Equivalence. LNCS 1784 (2000) 192–207
10. Boiten, E., Bujorianu, M.: Exploring UML refinement through unification. In Jürjens, J., Rumpe, B., France, R., Fernandez, E., eds.: Critical Systems Development with UML - Proceedings of the UML'03 workshop. Number TUM-I0323, Technische Universität München (September 2003) 47–62
11. Ducournau, R.: "Real world" as an argument for covariant specialization in programming and modeling. In: Proc. of the Workshops on Advances in OOIS. Volume 2426 of LNCS., London, UK, Springer (2002) 3–12
12. Gout, O.: Développement incrémental de spécifications orientées objets. PhD thesis, École des mines d'Alès, université de Montpellier 2 (december 2006)
13. Luong, H.V., Lambolais, T., Courbis, A.L.: Implementation of the conformance relation for incremental development of behavioural models. MODELS 2008,International Conference on Model Driven Engineering Languages and Systems (2008)