Preprints of the 18th IFAC World Congress
Milano (Italy) August 28 - September 2, 2011

Interoperability Analysis of Systems

Thomas Lambolais Anne-Lise Courbis Hong-Viet Luong Thanh-Liem Phan

LGI2P, École des mines d'Alès, France
(e-mail: firstname.familyname@mines-ales.fr).

Abstract: This work deals with the analysis of the behavioural interoperability of systems in a designing context. Systems to be analysed are modelled as UML architectures in terms of assembly of components, the behaviour of which are defined by State Machines. Two interoperability levels are identified: the absence of deadlock, and the preservation of the required services and usage protocols. The analysis starts in automatically transforming the behaviour of an architecture into a Labelled Transition System. A deadlock search can be then performed through model checking tools. Relations enabling components to be substituted without analysing again the whole system are identified, which leads to define a notion of component compatibility. This analysis technique checks if the interoperability of the new system is altered or preserved. Three cases are underlined: (i) the new system exhibits deadlocks: it is not interoperable any more, (ii) it is deadlock free, (iii) it is deadlock free and moreover conforms to the previous one. Results are illustrated through a case study.

Keywords: interoperability analysis, conformance, compatibility, architecture design.

1. INTRODUCTION

Our area of interest concerns system interoperability analysis and interoperable system design. Based on the conventional approach of modelling, we assume that system components are implemented according to standards they have to *conform to.* By this way, system implementations defined by assembling conformant components are expected to be interoperable. In the same way, when the system implementation has to be improved, we could think that the substitution of a component by a conformant one leads again to an interoperable system. If we refer to standardized definitions of conformance and interoperability (ISO/IEC 9646-1 (1991); ETSI EG 202 237 (2007)), these properties are unfortunately wrong. It appears that the conformance of a product to a specification is a necessary but not sufficient condition.

In this paper, our goal is to present a formal means to analyse the interoperability of systems described at first by UML architectures of components, and to propose complementary relations to conformance in order to guarantee the above properties. Our approach for conformance and interoperability analysis is based on Labelled Transition Systems (LTS). The system to be designed is defined by a set of interacting components the behaviour of which is defined by a State Machine. State Machines are automatically transformed into LTS for conformance analysis. Architecture descriptions are transformed into LTS for interoperability analysis. The analysis is performed at a high abstraction level: we consider interactions (exchange of messages or signals) existing between the system and its environment and between its components without taking into account data operations.

The paper is organised as follows. In section 2, we informally introduce concepts of conformance and interoperability and recall standardized definitions. Section 3 presents the conformance relation we have implemented to check component implementation expressed by State Machines. In section 4, we define a method to verify interoperability of systems. We point out that substitution of conform components do not guarantee interoperable systems. In section 5, we present relations that are appropriate to build up interoperable systems. Lastly, we present our future work. All sections are illustrated through an example.

2. INTEROPERABILITY AND CONFORMANCE DEFINITIONS

We refer to standardized definitions issued from Monkewich (2006), Wiles (2003) and Lynskey (2003) about system interoperability and conformance of its components to given standards.

2.1 Interoperability

The system under study is considered as a set of parts called entities, devices or components. According to Lynskey (2003), a system is interoperable if, under a given set of conditions, its devices are able to successfully establish, sustain, and if necessary, tear down a link while maintaining a certain level of performance.

If the notion of "level of performance" could be discussed, it at least requires the system to be able to run without blocking. So, in a restricted point of view, as defined by Baldoni et al. (2009), a set of parts is *interoperable* when it is "stuck-free", i.e., whatever point of interaction may be reached, communication will not be blocked, and each part will reach one of its final states.

2.2 Conformance

A device is said to be conformant, or compliant, to a standard if it has properly implemented all the mandatory portions of that standard. We refer to ISO9646 ISO/IEC 9646-1 (1991) conformance testing methodology, or IEEE 802.3. Hence, the formal definition of conformance we adopt translates the following property: any test that the specification *must* accept, *must* also be accepted by the implementation. Stated otherwise, any test that the implementation *may* refuse, *may* also be refused by the specification. Conformance has been formalized by Brinksma and Scollo (1986) and Tretmans (1999).

According to this definition, a conformant implementation may implement more functions or services than its specification, but may also implement less services when optional services are omitted. Hence, the composition of conformant components may lead to non interoperable systems. Conformance is a required condition, but it does not guarantee the system to be interoperable.

3. STATE MACHINE CONFORMANCE

We have been interested in developing a method to verify the conformance of behavioural models expressed by UML State Machines, based on work of Brinksma and Scollo (1986) and Tretmans (1999). The referring model represents a specification or a standard. It is expressed at a high abstraction level, using the state machine formalism. One condition has to be respected: interfaces of the implementation and the specification have to be the same.

The verification of conformance requires a formal support for reasoning, underlining properties to be verified (sequences of events, synchronization, parallelism, deadlocks and livelocks) and hiding those which are not significant, such as data transformations or timing aspects.

In the following subsections, we give definitions of LTS formalism and an overview of the conformance relation we have implemented over LTSs. We thus give rules allowing state machines to be automatically transformed into LTS.

3.1 Main features of LTS

We give some useful definitions to understand our approach and illustrations. For more details, refer to Milner (1999). Let $\mathcal{P}$ be a set of states or process names, and Act a set of names of actions, with $Act = \mathcal{L} \cup \{\tau\}$, where $\mathcal{L}$ is the set a visible actions and τ a silent action.

Definition 1. (LTS, Milner (1989)). A LTS $\langle P, A, \rightarrow, p\rangle$ is a tuple of a non empty set $P \subseteq \mathcal{P}$ of states (or processes), a set $A \subseteq Act$ of action names, a relation of transitions $\rightarrow \subseteq P \times A \times P$, and an initial state $p \in P$.

We designate by p the LTS $\langle P, A, \rightarrow, p\rangle$. Let us note that the term *actions* used in the LTS formalism may refer to UML concepts of actions, activities and events. It designates a visible (public) port enabling component synchronizations.

We designate by $Tr(p)$ the set of traces of the LTS p, where a trace is a possibly incomplete sequence of observable actions. The set of states which can be reached from p after a trace σ is denoted by p after σ. $Out(p, \sigma)$ is the set of observable actions of p after the trace σ.

Definition 2. (Acceptance set, Leduc (1992)). The acceptance set of p after a trace σ, noted $Acc(p, \sigma)$, is defined by $Acc(p, \sigma) = \{X \mid \exists p' \in p \text{ after } \sigma.\ X = Out(p', \varepsilon)\}$. It is the set of action sets representing actions that *must* be accepted by p after the trace σ.

This notion is crucial for the conformance relation because it defines for any situation (trace concept) what the component is supposed to accept (actions or events), even if it is non deterministic. It captures the concepts of mandatory and optional actions. For instance, the acceptance set $\{\{a, b\}, \{b\}\}$ (after a trace σ) means that actions a and b can be accepted by the component after σ, and that b is a mandatory action but action a may be refused after σ.

3.2 The conformance relation

Definition 3. (Conformance, Brinksma and Scollo (1986)). $q \ \mathsf{conf}\ p$ if $\forall \sigma \in Tr(p)$, $Acc(q, \sigma) \subset\subset Acc(p, \sigma)$.

The relation $\subset\subset$ is an inclusion relation defined over sets of sets: $A \subset\subset B$ if, for all a in A, there exists b in B such that $b \subseteq a$.

If $q \ \mathsf{conf}\ p$, the process q is more deterministic than process p.

No algorithm of formal checking of conformance had been proposed. The algorithm we have developed to check conformance is based on the notion of process merging. The theorem on which it is based can be found in Luong et al. (2008). It has been implemented in a JAVA tool, called IDCM (Incremental Development of Conformant Models) which provides, as we will see in the following, other comparison relations.

3.3 Transformation of State Machines into LTS

We now consider a component described in UML, whose provided and required interfaces are known. It is synchronized with its environment through methods defined in interfaces. Its behaviour is defined by a State Machine the evolution of which depends on call events, signal events, time events, change events and complete events. Actions occurring on transitions or activities associated with states can be internally performed (using private methods) or delegated to other components using methods defined in its required interfaces. The internal actions are translated using the τ notation of LTS. Other actions are labelled in the LTS using their UML name. UML guards are represented from an abstract point of view, using again the τ notation. It is interpreted as an internal decision which is not under control of the environment. Time is also modelled by internal transitions τ. It is interpreted by the fact that, whatever the delay is, the component will eventually reach a given state.

The transformation of UML State Machines into LTS is defined by compositional rules. Every UML state is transformed into a LTS state. Every output transition is

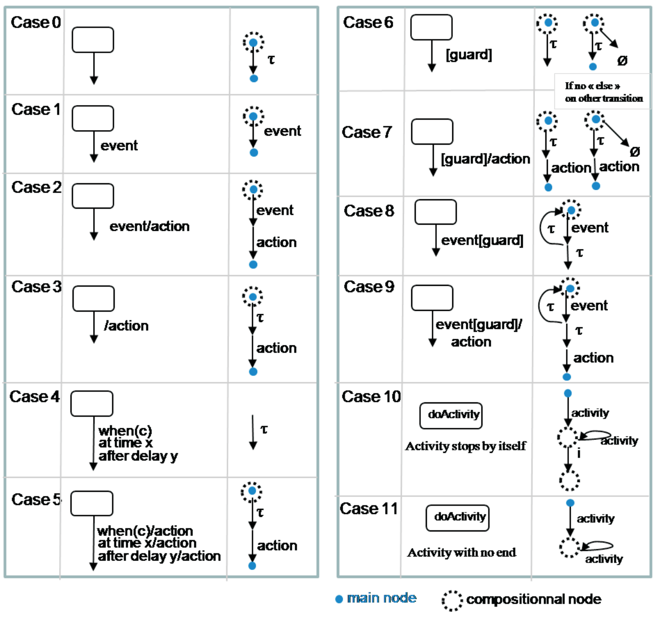

Fig. 1. Rules to transform State Machines into LTS

transformed according to a pattern defined in Fig. 1. Rules are compositional because transitions with a common input state are transformed into LTS transitions starting from the state labelled as compositional (see Fig. 1). Let us comment on case 9 dealing with a transition of type $event[guard]/action$. It is transformed into a LTS where the main node corresponds with the input state of the UML transition. The τ transitions mean that the system may stay in the same state (if that guard is false) or may change of state (if that guard is true). In this latter case, the reached state must thus accept *action*. If there are other transitions leaving the UML state, they are transformed by applying again one of the defined cases starting from the composition node of the already built up LTS graph.

The transformation has been implemented in Java in the TopCased environment (Farail et al. (2006)). Results of transformation are written in the LTS ASCII-based format of CADP (Construction and Analysis of Distributed Processes, Garavel et al. (2007)), in order to be analysed in the CADP toolbox.

3.4 Illustration of component modelling and conformance checking

Let us consider a specification of a component named *TaskManagement* whose goal is to perform a task received from its environment and to send it back. The task can be processed internally by *TaskManagement* or sent to an outsourced component. The condition of this choice is not defined by the specification. The component and its interfaces can be seen in Fig. 6 representing the context in which the component will be used. Its expected behaviour is defined by a UML State Machine given in Fig. 2.

This specification is implemented in a component whose interfaces are the same. They differ in behavioural aspects (see State Machine in Fig. 3). In this implementation, the choice of internal or external processing is done according

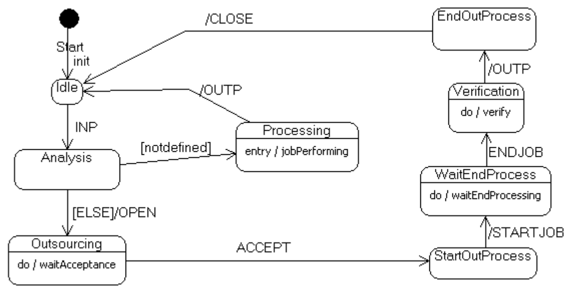

Fig. 2. TMSpec$_{SM}$: TaskManagement specification State Machine

to the criteria *easyJob* that is supposed to be evaluated by the component itself in the *jobAnalysis* activity.

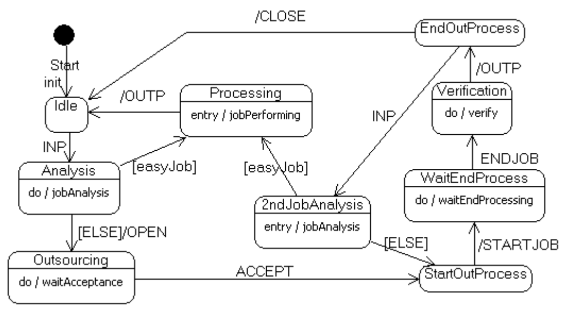

Fig. 3. TMImp$_{SM}$: TaskManagement implementation State Machine

The two State Machines are transformed into LTS by IDCM (see Fig. 4, where action i corresponds to τ). The conformance of the implementation to its specification is verified using IDCM and concludes that "TMImp$_{LTS}$ conf TMSpec$_{LTS}$".

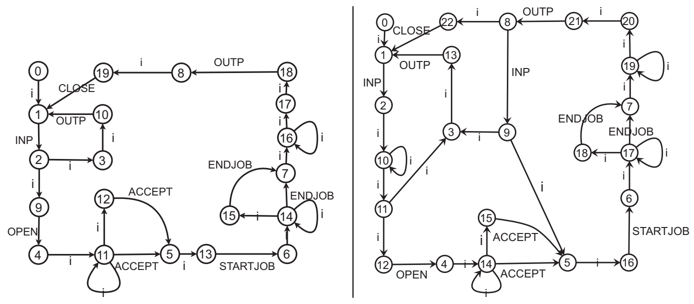

Fig. 4. TMSpec$_{LTS}$ and TMImp$_{LTS}$.

4. INTEROPERABILITY ANALYSIS

We now deal with the interoperability analysis of an assembly of components. We give formal definitions of deadlock and interoperability of systems. Our goal is at first to analyse the interoperability of specifications, and then, the interoperability of a system, the components of which conform to specifications. Here, we consider specifications like standards, which should be conceived to work together. At first, it is necessary to verify that an assembly of such specifications is interoperable. When the interoperability of the real system is considered, this will raise up the problem of component substitutability. This will be discussed in last subsection.

4.1 Formal definition of interoperability

Definition 4. (Deadlock). A process defined by a LTS p deadlocks after a trace σ if $Acc(p, \sigma) = \varnothing$.

In order to distinguish a final state from a deadlock, we introduce the concept of stuck free process.

Definition 5. (Stuck free). A system s of components p_1, p_2, ..., p_n is stuck free, if all its deadlocks correspond to common final states of p_1, p_2, ..., and p_n: for any trace σ of s corresponding to subtraces σ_1 of p_1, ..., σ_n of p_n, $Acc(s, \sigma) = \varnothing \Rightarrow Acc(p_1, \sigma_1) = \cdots = Acc(p_n, \sigma_n) = \varnothing$.

A system is interoperable if it is stuck free.

4.2 Architecture specification: modelling and analysis

Let us consider TaskManagement specification given in Fig. 2. TaskManagement is designed to work with an outsourced component, named TaskTreatment. TaskTreatment specification State Machine is given in Fig. 5.

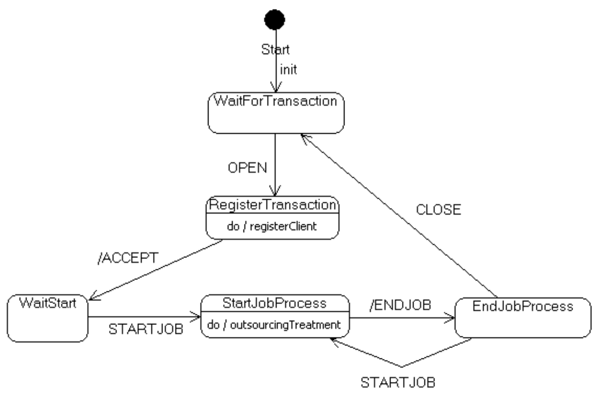

Fig. 5. TTSpec$_{SM}$: TaskTreatment specification State Machine

Let us consider the architecture composed of these two components (Fig. 6). Its provided interface offers one service: INP representing an input task. Its required interface delivers the performed task through OUTP method. The interoperability of $Arch_0$ is expected. Hereafter, we aim at verifying this formally.

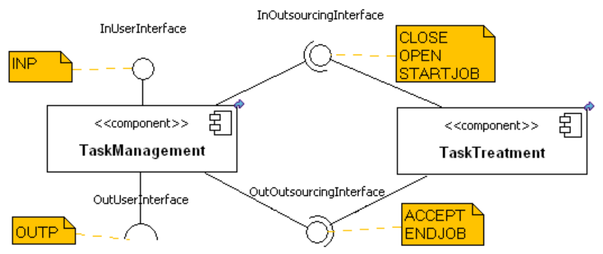

Fig. 6. Architecture $Arch_0$ composed with TaskManagement and TaskTreatment

Analysing interoperability requires a formal semantics of architectures. We chose the language LOTOS NT (Sighireanu et al. (2004)), and map UML components onto LOTOS NT processes. The LOTOS NT language is translated into LOTOS language whose operational semantics is given in LTS. The example of Fig. 7 illustrates the LOTOS description corresponding to the architecture $Arch_0$. Despite UML and LOTOS concepts are different, some correspondences can be done: UML components are associated with LOTOS NT processes ; the different types of UML events correspond to LOTOS actions ; component connections correspond to LOTOS parallel composition operators. In the example, the two components are modeled by LTS processes, TMSpec$_{LTS}$ and TTSpec$_{LTS}$, the interfaces of which are defined in terms of lists of actions ; the syntax |[OPEN, START, CLOSE, ACCEPT, ENDJOB]| means that the two processes are synchronized on all actions of the list. A way to model asynchronous communication would have been to add another component modelling a kind of communication channel.

```
specification Arch0[INP,OUTP]:noexit
behaviour
hide OPEN,START,CLOSE,ACCEPT,ENDJOB in
     (TMSpecLTS[INP,OUTP,OPEN,START,CLOSE,ACCEPT,ENDJOB]
       |[OPEN,START,CLOSE,ACCEPT,ENDJOB]|
      TTSpecLTS[OPEN,START,CLOSE,ACCEPT,ENDJOB] )
endspec
```

Fig. 7. LOTOS description of architecture $Arch_0$

Since the two LTS TMSpec$_{LTS}$ and TTSpec$_{LTS}$ are automatically generated by IDCM State Machine transformation, the CADP toolbox can generate the LTS of the architecture according to LOTOS semantics. The result is a graph made up of about 70 states and 150 transitions.

The analysis performed with CADP concludes that this architecture specification is deadlock free.

4.3 Architecture realization: modelling and analysis

Let us now consider again architecture of Fig. 6, the components of which are now implementation models. We call $Arch_1$ this new architecture. The State Machine implementation model of TaskManagement is given in Fig. 3. To simplify, we assume that TaskTreatment implementation has the same State Machine as its specification, given in Fig. 5. State Machine of TaskManagement implementation is translated into LTS, named TMImp$_{LTS}$.

As previously, a LOTOS NT description is associated with this architecture. It is the same as the architecture specification, except that specification models are replaced by implementation models. The generated LTS of the realization architecture is made up of about 125 states and 270 transitions.

Analysis of the implementation architecture through CADP toolbox concludes that this architecture is deadlocked. So, the architecture realization is not interoperable.

One sequence of events leading to a deadlock, given by the tool, is: INP; OPEN; ACCEPT; STARTJOB; ENDJOB; OUTP; INP; OUTP; INP. Its analysis points out that first task received by TaskManagement is sent to TaskTreatment. The second one is internally processed by TaskManagement. The last one is sent to TaskTreatment by requesting an OPEN transaction but the request fails. It is explained by the fact that TaskTreatment is in state 2ndJobAnalysis and is waiting for a transaction closing. It can not receive any OPEN order, which is the reason of deadlock. TaskManagement is wrong: it does not emit a CLOSE action at the end of the first outsourced task. Such a behaviour is possible although TaskManagement implementation conforms TaskManagement specification. Indeed, every sequence of actions that the specification must do, must be done by the implementation. But the implementation may also do more. It does not prevent the implementation from receiving new jobs (action INP) in the EndOutProcess state. Such a new trace is not under the scope of the conformance relation. The implementation must indeed do a CLOSE action like before, but only if the new INP has not been received. So, the fact that component implementations conform their specifications does not guarantee the interoperability of the new system. The analysis of the error enables us to correct TaskManagement State Machine: an action CLOSE is added from state 2ndJobAnalysis to state Processing (see Fig. 8). With this new implementation, called TMImpC$_{SM}$, the architecture is deadlock free.

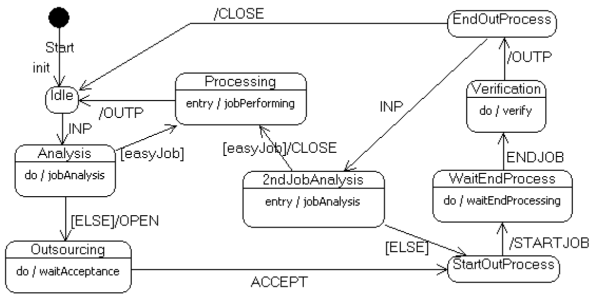

Fig. 8. TMImpC$_{SM}$: Corrected TaskManagement implementation State Machine

This case study raises two problems. Which conformance relation guarantees the interoperability of systems? Is there another way to guarantee the interoperability of a system in which a component is substituted by another one, without computing the behaviour of the whole system?

5. BUILD-UP OF INTEROPERABLE SYSTEMS

We have studied how to analyse a system in order to check its interoperability. In this section, the goal is to identify relations between implementations and interoperable specifications that could guarantee the interoperability of the system. Moreover, during a redesign process, it is necessary to know not only if the new system is still interoperable, but also if services offered by the old system are preserved. Extension relations are good supports to doing that.

5.1 Substitutability

If a component C has to be replaced by a component R, we must not only check that R must do what C must do (what conformance checks), but also that R does not offer any new observable behaviors. Indeed, R must be able to provide what C provides, and R must not require any new service which may interfere with its environment. That is the reason why conformance is not enough. Relations which satisfy such substitutability properties in any context are congruence relations. Congruence relation defined over the the conformance relation are cext and cred.

Definition 6. (cred, cext Brinksma and Scollo (1986)). Let p and q be two LTS,

q cred $p \Leftrightarrow$ for any context $C[.], C[q]$ red $C[p]$,
q cext $p \Leftrightarrow$ for any context $C[.], C[q]$ ext $C[p]$.

The context refers to the set of components interacting with the component under analysis. Relations red and ext are defined as conformance relations combined with trace inclusion and trace extension. These relations have been implemented in IDCM, as well as cext and cred.

For instance, the State Machine TMImp2$_{SM}$ presented in Fig. 9 is such that TMImp2$_{LTS}$ cred TMSpec$_{LTS}$: indeed, TMImp2$_{SM}$ has no more traces than TMSpec$_{LTS}$ and conforms to TMSpec$_{LTS}$. The cred relation also checks that all stable states are preserved. A state is *stable* if the machine may not leave it by itself (after a complete event, change event or time event for instance).

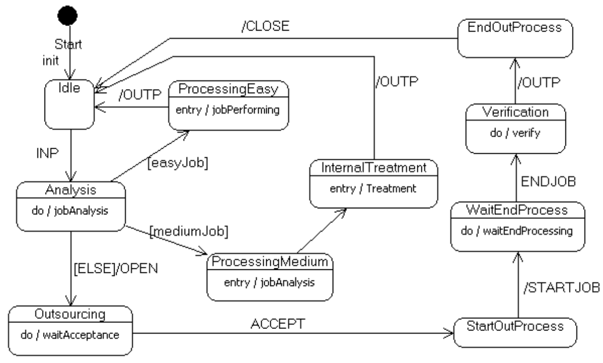

Fig. 9. TMImp2$_{SM}$: Second TaskManagement implementation State Machine

Hence, the architecture of Fig. 6 where TaskManagement is replaced by a component whose State Machine is TMImp2$_{SM}$, is still interoperable. We call Arch2 this architecture.

Congruence relations are appropriate to evaluate interoperability. Nevertheless, they are strong and we would want to define a weaker relation. For this reason, we introduce the notion of compatibility.

5.2 Compatibility

If the new component is not in relation with the former one by a known congruence relation, as it is the case for TMImpC$_{SM}$ (Fig. 8), it is possible to verify its compatibility with its context. Since the architecture is translated into a LOTOS NT expression, all interactions are expressed by binary parallel composition operators. That means it is always possible to consider the context as a set of components that can be modelled by a unique LTS. By this way, the notion of compatibility of a component according to its context, is expressed by a compatibility relation between two LTS.

Definition 7. (LTS compatibility). Two LTS p and q are compatible on a set of actions S, written p comp$_S$ q, if for any trace $\sigma \in S^*$, $Acc(p/S, \sigma)$ comp $Acc(q/S, \sigma)$, where

p/S is the LTS obtained from p by hiding any action not in S.

Definition 8. (Set compatibility). Two sets of sets A and B are compatible, written $A \ \mathsf{comp}\ B$, if $\forall x \in A, \forall y \in B, x \cap y \neq \varnothing$. Note that, due to indeterminism, a set may be incompatible with itself.

Theorem 1. If $p \ \mathsf{comp}_S\ q$, then $p|[S]|q$ is stuck free.

Indeed, after any trace $\sigma \in S^*$, $Acc(p/S,\sigma) \neq \varnothing$ and $Acc(p/S,\sigma)\ \mathsf{comp}\ Acc(q/S,\sigma) \Rightarrow Acc(q/S,\sigma) \neq \varnothing$. So, for any trace σ_p and σ_q such that $p' \in p \text{ after } \sigma_p \Leftrightarrow p' \in p/S \text{ after } \sigma$ and $q' \in q \text{ after } \sigma_q \Leftrightarrow q' \in q/S \text{ after } \sigma$, $Acc(p,\sigma_p) \neq \varnothing \Rightarrow Acc(q,\sigma_q) \neq \varnothing$.

For instance, we can demonstrate that TMImpC$_{LTS}$ is compatible with the context of an architecture composed with TTImp$_{LTS}$. Actions which are not hidden are those belonging to InOutSourcing and OutOutsourcing interfaces: $S = \{$OPEN, CLOSE, STARTJOB, ACCEPT, ENDJOB$\}$. Their acceptance sets are the same (and are singletons, so compatible with themselves), except after trace $t =$ OPEN; ACCEPT; STARTJOB; ENDJOB:
$Acc(\text{TMImpC}_{LTS}/S,t) = \{\{\text{CLOSE}\}, \{\text{STARTJOB}\}, \{\text{CLOSE}, \text{STARTJOB}\}\}$, and
$Acc(\text{TTImp}_{LTS}/S,t) = \{\{\text{CLOSE}, \text{STARTJOB}\}\}$. These two acceptance sets are compatible. So, the two components are compatible.

5.3 Interoperability performance level

We have defined relations to check if a system is deadlock free. That is the first level of performance required for the interoperability. In case of redesigning, we could expect more from an interoperability relation: under the same conditions, the system should offer the same services as the previous one. This is a property checked by the conformance relation. Having defined a referring architecture, any new architecture can be checked according to it, from an external point of view (the user point of view) by hiding internal interactions. For example, the conformance of the two architectures $Arch_1$ and $Arch_2$ according to $Arch_0$ can be proved, by hiding InOutsourcing and OutOutsourcing interfaces.

6. CONCLUSION

Based on standard definitions, we have formally defined interoperability of systems defined by UML architectures, where component behaviours are described by State Machines. We have implemented transformations from UML to LTS, and defined relations to check interoperability. We have shown that replacing in a system a component by another conform one, does not guarantee the system interoperability. Interoperability has thus to be checked following two ways: analysing the whole behaviour of the architecture or comparing the new substituted component according to its first implementation or its context. All relations useful for interoperability and conformance checking have been implemented in our prototype IDCM, developed in the TopCased environment.

An interesting application of this work is to develop an automatic procedure to select components from a library. Having defined an architecture specification, interoperable and conform architectures could be automatically built-up. One feature to be developed is an optimizing unit that could choose from candidate architectures the most promising. Criteria of optimization could be: resource consuming (number of components of the implementation), the use of critical components, the number of interactions, the use of reliable components, etc.

REFERENCES

Baldoni, M., Baroglio, C., Chopra, A.K., Desai, N., Patti, V., and Singh, M.P. (2009). Choice, interoperability, and conformance in interaction protocols and service choreographies. In S. Decker Sichman and Castelfranchi (eds.), *8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009).* Budapest, Hungary.

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

ETSI EG 202 237 (2007). Generic approach to interoperability testing.

Farail, P., Gaufillet, P., Canals, A., Camus, C.L., Sciamma, D., Michel, P., Crégut, X., and Pantel, M. (2006). The TOPCASED project: a toolkit in open source for critical aeronautic systems design. *Embedded Real Time Software (ERTS).*

Garavel, H., Lang, F., Mateescu, R., and Serwe, W. (2007). CADP 2006: A Toolbox for the Construction and Analysis of Distributed Processes. In *Proceedings of the 19th International Conference on Computer Aided Verification CAV 2007.*

ISO/IEC 9646-1 (1991). Information technology – Open Systems Interconnection – Conformance testing methodology and framework – Part 1: General concepts.

Leduc, G. (1992). A framework based on implementation relations for implementing LOTOS specifications. *Computer Networks and ISDN Systems*, 25(1), 23—41.

Luong, H., Lambolais, T., and Courbis, A. (2008). Implementation of the conformance relation for incremental development of behavioural models. In K. Czarnecki (ed.), *MoDELS 2008*, volume 5301/2009 of *LNCS*, 356–370. Springer-Verlag.

Lynskey, E. (2003). Importance of last mile interoperability. OptiCom 2003, 1st Internation Workshop on Community Networks and FTTH/P/x.

Milner, R. (1989). *Communication and concurrency.* Prentice-Hall, Inc.

Milner, R. (1999). *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, 1st edition.

Monkewich, O. (2006). Conformance and interoperability testing tutorial. UIT-T SG17, Telecommunication Languages and Softwares. Second Informal Workshop. Switzerland, Genève.

Sighireanu, M., Chaudet, C., Garavel, H., Herbert, M., Mateescu, R., and Vivien, B. (2004). LOTOS NT User Manual. *INRIA, june.*

Tretmans, J. (1999). Testing concurrent systems: A formal approach. In S.M. Jos C.M. Baeten (ed.), *CONCUR99 Concurrency Theory*, volume 1664 of *LNCS*, 46–65. Springer-Verlag, Berlin Heidelberg.

Wiles, A. (2003). Relevance of conformance testing for interoperability testing. ACATS ATS-CONF. Stuttgart.