Cooperation Between the B Method and the Automata Theory to Check the Component Interoperability

Samir Chouali [1]

Laboratoire d'Informatique de l'Universite de Franche-Comte - LIFC FRE CNRS 2661
16, route de Gray - 25030 Besanon cedex, France

Abstract

Component interoperability is one of the essential issues in the component based development, since it allows the composition of reusable heterogenous components developed by different people. In this paper, we propose an approach to formally verify component interoperability at signature, semantics, and protocol levels. It is based on the use of the B formal method for specifying component interfaces and finite transition systems for specifying component protocols. The verification is done with the B theorem prover and the verification of the simulation relation between transition systems. This approach allows to decide whether two components can interoperate if assembled together and whether a component can be replaced by another component.

Keywords: Component interoperability, compatibility, substitutability, B method, verification.

1 Introduction

The component based development is widely used in software engineering. Its aim is to develop software systems by assembling a collection of pieces that are independently produced. These pieces are called components. The advantages of this approach are the reusability of software components which involves a reduction of the development cost, and the flexibility of developed systems.

A component is a unit of composition with contractually specified interfaces and explicit dependencies [22]. An interface describes the services offered or required by a component without disclosing the component implementation. It is the only access to the information of a component. The offered services by a component are described by an offered interface and the required services are described by a used interface.

The component interoperability is an essential issue in the component based development. The success of applying the component based approach depends

[1] Email: chouali@lifc.univ-fcomte.fr

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

on the interoperability of the connected components. The interoperability can be defined as the ability of two or more entities to communicate and cooperate despite differences in their implementation language, the execution environment, or the model abstraction [11,24]. In the verification of component interoperability, it is necessary to consider two cases:

- verification of component compatibility: verify whether two software components can be related.
- verification of component substitutability: verify whether a defective component can be replaced by another component.

So, the verification of component interoperability involves the verification of the interface compatibility or interface equivalence on the components that will be connected or the component that will be replaced.

The specification of the interfaces plays an important role in the verification of their compatibility. Most current interface modelling languages (IDLs), used in several component oriented platforms like JavaBeans [21], CORBA [16], or COM [14], are limited for expressing signature (operation names, types, parameters) information. They provide insufficient information about component behaviors. Hence, one cannot insure trust in component based systems.

The design by contract approach proposed by B. Meyer [13] is the first evolution toward a specification approach that proposes a way to insure trust in components. B. Meyer proposes to annotate interface specifications with assertions that provide pre and postconditions of the operations. Several other works [9,1] have proposed to enhance component interfaces by providing information at signature, semantics (the meaning of operations) and protocol (order in which the operations of a component are called) levels. Despite these enhancements in the interface specification, we believe that there is not enough information in component interfaces to perform a formal verification of the interface compatibility.

In this paper we deal with the formal specification of the interfaces and the verification of their compatibility at the signature, semantics, and protocol levels. We propose to specify interfaces with the B method and to enhance them by the specification of protocols using the automata theory language [18]. The B method allows an abstract specification of systems to be developed step by step using the refinement until the implementation. Therefore we consider that the B specification of an interface is an abstraction of a component implementation.

The B theorem prover is used to verify a step of the interface compatibility. We also exploit the B specifications of the interfaces and their protocol specifications in order to model the behavior of software components with transition systems [4]. Thus, we verify the simulation relation between the transition systems, which is the final step of the verification of the interface compatibility.

In the following section we give an overview of the B method. In Section 3 we present the specification of component interfaces and illustrate it by the example of the car wiping system. In Section 4 we present the verification approach of the component interoperability. In Section 5 we present related work. In section 6 we terminate the paper by a conclusion.

2 The B method

The B method [2] is a formal software development approach allowing to develop software for critical systems. It covers the entire development process from abstract specifications to an implementation. The B method is based on the set theory. The basic building block of a B specification is the abstract machine that is similar to a module or a class in an object oriented development. A B specification is composed of one or many abstract machines (examples of B machines are given in Section 3), each of them describes a set of variables, invariance properties (also called safety properties) relating to these variables, an initialization which is a predicate that initializes the variables, and a list of operations. The specification of an operation consists of a precondition part and a body part. The precondition expresses the requirement that must be met whenever the operation is called. The body expresses what the operation achieves, it is expressed with a generalized substitution. The states of a specified system are only modifiable by operations that must preserve an invariant.

With the B method, a system is developed by refinement. The refinement is used to transform step by step an abstract specification into a concrete representation. At each refinement step, you have to prove that the refined specification is correct with its abstraction. Therefore the implementation must refine its abstract specification. The verification with the B method is automated. The B theorem prover, Atelier B [20], allows the verification of invariance properties and the refinement relation.

3 Specification of component interfaces

Our goal is to propose an approach to specify component interfaces in order to verify the interface compatibility. This work takes place in the context where components are specified as black boxes, then deployed without knowing details of their implementation. In this context, the specification of component interfaces plays an important role because it is the only description of the component.

We propose to specify components interfaces with the B method and augment the interface specifications with protocol specifications that must be respected by the order of the operation calls. We apply this approach to a case study.

3.1 Specification structure

Traditional approaches of interface specification provide signature of operations and their pre and postconditions. In many cases there are not sufficient information in interfaces specifications in order to detect the problems of components

3.1.1 Interface Specification with the B method

A B specification of an interface is composed of the following information:

- The name of the B machine: it is the name of the associated interface.
- The variables of the interface: the set of variables used in the operation definition.
- The initialisation: the initial value of the variables.

- The invariant: describes the property that a component must satisfy in each state.
- The operations: each operation is specified by a precondition and a body which expresses the transformation performed on states by an operation.

3.1.2 Protocol specification

In the B method, it is not possible to specify a particular order of operation calls. Therefore we use the regular language to specify component protocols. We specify a protocol as a set of finite words that are made up over an alphabet which is a set of the called operation names.

A protocol is described by the following formula:
$(((operation\_name_1).(operation\_name'_1)....)^* + ...$
$+((operation\_name_i).(operation\_name'_i)....)^* + ...)$, using the following operators:

- $''.''$ expresses the sequencing in the operation calls. For example $(operation\_name_1 \ . \ operation\_name_2)$ means that the operation $name_1$ will be called before the operation $name_2$,
- $''+''$ expresses the choice. For example $(operation\_name_1 \ + \ operation\_name_2)$ means that either the operation $name_1$ or the operation $name_2$ will be called,
- $''*''$ is used to express the repetition. For example $(operation\_name_1 \ . \ operation\_name_2)^*$ means that the operation $name_1$ and the operation $name_2$ will be called a finite number of times.

3.2 Case Study: a car windscreen wiping system

We illustrate our approach by considering a car windscreen wiping system [12]. This system receives messages from an environment, a car driver, and activates the windscreen wiper. It has two operational modes:

- manual: the car driver selects the speed for the windscreen wiper. Then the wiping system receives messages from the car driver and sends messages in order to activate the windscreen wiper with the selected speed,
- automatic: the wiping system is activated by the car driver and the selection of the speed for the windscreen wiper is done automatically according to the quantity of the detected rain.

The wiping system can send messages to activate the windscreen wiper with three different speeds: the first speed, the intermediate speed and the second speed.

In order to construct this system, we dispose of three components provided by different software designers. The components are:

- the control lever CLever component,
- the windscreen wiper WWiper component,
- the rain sensor RSensor component.

3.2.1 The CLever component

It allows to detect the mode of the wiping system selected by the car driver, automatic or manual. In the first mode, the component CLever sends a message to activate the automatic mode for the wiping system. In the second mode, the car driver can select two speeds for the windscreen wiper:

- speed1: the component CLever sends a message to the environment in order to activate the first speed for the windscreen wiper,
- speed2: the component CLever sends a message to the environment in order to activate the second speed for the windscreen wiper.

Remark 3.1 Note that the component CLever does not offer the possibility to select the intermediate speed for the wiping system.

The component CLever has three interfaces: the offered interfaces *OIManual* and *OIAutoCL*, and the used interface *UICLever*. We only describe the offered interface *OIManual*. The B specification of this interface is given in figure 1.

MACHINE OIManual
SETS
$CL\_Position = \{sp_1, sp_2, stop\}$;
VARIABLES
cl_pos
INVARIANT
$cl\_pos \in CL\_Position \qquad (I_o)$
INITIALISATION
$cl\_pos := stop$
OPERATIONS
$Cl\_Speed1 \triangleq$ **PRE** $(cl\_pos = sp_2 \vee cl\_pos = stop)$
THEN $cl\_pos := sp_1$ **END**;
$Cl\_Speed2 \triangleq$ **PRE** $cl\_pos = sp_1$
THEN $cl\_pos := sp_2$ **END**;
$Cl\_Stop \triangleq$ **PRE** $cl\_pos = sp_1$
THEN $cl\_pos := stop$ **END**

Fig. 1. B machine corresponding to the interface *OIManual* of the component CLever

The operations defined in the interface *OIManual* are:

- *Cl_Speed*1: the car driver activates the first speed for the wiping system.
- *Cl_Speed*2: the driver activates the second speed for the wiping system.
- *Cl_Stop*: the driver stops the wiping system.

The protocol that must be respected by the operation calls in the interface *OIManual* is:

$(Cl\_Speed1.(Cl\_Speed2.Cl\_Speed1)^*.Cl\_Stop)^*$.

3.2.2 The WWiper component

It receives messages from the environment and activates (with the first, the second or the intermediate speed) or stops the windscreen wiper.

This component has three interfaces, the used interfaces *UIManual* and *UIAutoWW*, and the offered interface *OIWWiper*. We only describe the used interface *UIManual*. The B specification of this interface is given in figure.2.

MACHINE UIManual
SETS
$CL\_Position = \{sp_1, sp_2, sp_i, stop\}$;
VARIABLES
cl_pos
INVARIANT
$cl\_pos \in CL\_Position \qquad (I_u)$
INITIALISATION
$cl\_pos := stop$
OPERATIONS
$Cl\_Speed1 \triangleq$ **PRE** $(cl\_pos = sp_2 \vee cl\_pos = sp_i \vee cl\_pos = stop)$
THEN $cl\_pos := sp_1$ **END**;
. $Cl\_Speed2 \triangleq$ **PRE** $cl\_pos = sp_1$
THEN $cl\_pos := sp_1$ **END**;
$Cl\_Speedi \triangleq$ **PRE** $cl\_pos = stop$
THEN $cl\_pos := sp_i$
$Cl\_Stop \triangleq$ **PRE** $(cl\_pos = sp_1 \vee cl\_pos = sp_i)$
THEN $cl\_pos := stop$ **END**

Fig. 2. B machine corresponding to the interface *UIManual* of the component WWiper

The operations defined in the interface *UIManual* are:

- *Cl_Stop*, *Cl_Speed*1, *Cl_Speed*2: these operations have the same description as those described in the interface *OIManual*,
- *Cl_Speedi*: the driver activates the intermediate speed for the wiping system.

The protocol that must be respected by the operation calls in this interface is:
$((Cl\_Speedi.Cl\_Stop)^*.$
$(Cl\_Speed1.(Cl\_Speed2.Cl\_Speed1)^*.Cl\_Stop)^*.$
$(Cl\_Speedi.Cl\_Stop)^*)$.

3.2.3 The RSensor component

It is operational in the automatic mode of the wiping system. It allows to detect the quantity of rain and to send messages to its environment in order to activate the windscreen wiper with the appropriate speed. it can detect three cases: there is no rain, there is little rain and there is big rain. This component is described by the offered interface *OIAutoRS* and the used interface *UIAutoRS*.

3.2.4 The wiping system

The architecture of the wiping system using these three components is proposed in figure 3. It is described using UML 2 notations [8].

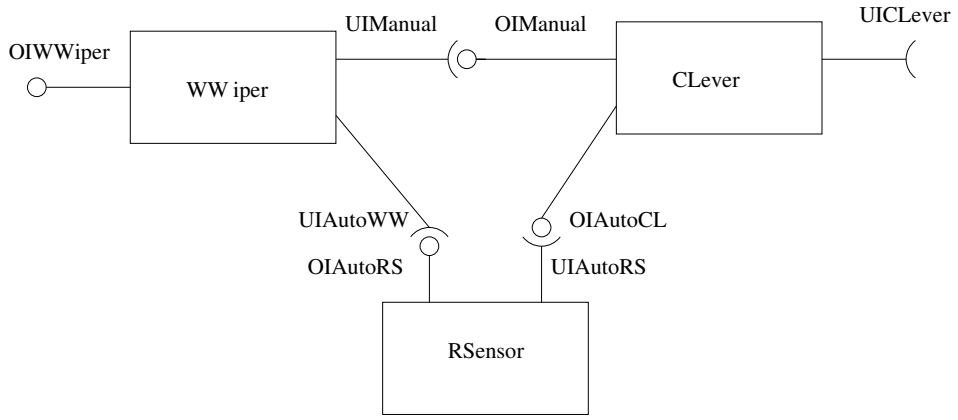

Fig. 3. Component architecture of the wiping system

4 Verification of component interoperability

In order to verify the component interoperability, it is necessary to treat two cases:

- verification of component compatibility: verify whether two software components can be related.
- verification of component substitutability: verify whether a component can be replaced by another component.

4.1 Verification of component compatibility

To determine whether two components can interoperate, we propose a compatibility verification between offered and used interfaces because components interoperate via these interfaces.

We use the B theorem prover, and the operational semantics of the interface specifications (transition systems) of offered and used interfaces to perform the verification. We split the verification approach into two steps:

- Invariant verification: since we define an invariant in the interfaces, we must verify that the invariant provided in the used interface is satisfied by the specification of the appropriate offered interface.
- Verification of interface compatibility at signature, semantics and protocol levels between an offered and used interfaces: to verify that required services described by a used interface are compatible with offered services described by an offered interface, it is necessary to verify the compatibility between these interfaces. Therefore, we verify that the component behavior described by an offered interface simulates the component behavior described by the corresponding used interface.

We illustrate our verification approach by checking whether the component CLever interoperates with the component WWiper via the interfaces *OIManual* and *UIManual*.

4.1.1 Invariant verification

In order to insure that the component CLever can interoperate with the component WWiper via the offered interface *OIManual* and the used interface *UIManual*, we must prove that the operations specified in the interface *OIManual* satisfy the invariant I_u specified in the interface *UIManual*. Since we specify interfaces using the B method, we use the B theorem prover to perform this verification. The verification will be performed on a new B specification obtained by modifying the *OIManual* specification and taking into account information provided by *UIManual*. So, we describe below two steps allowing to construct a B specification where the invariant I_u is specified and also the operations of the offered interface *OIManual* are specified. This new specification is obtained as follows:

- We modify the invariant of the interface *OIManual*: The invariant of the new specification is $I_o \wedge I_u$, where I_o is the invariant of the interface *OIManual*.
- We add in the specification of *OIManual* variables and sets of *UIManual* which are not defined in *OIManual*. If there exists a set S which is defined in the interface *OIManual* and in the interface *UIManual* with more variable values, then we replace the set S in *OIManual* by the set S of *UIManual* (see the example below). Finally, initialize the new variables defined in the specification of *OIManual*.

After carrying out these two steps, we obtain a new B specification presented in figure 4 in which the value sp_i has been added in the set $CL\_Position$. We did not change the invariant of the interface *OIManual* because this interface has the same invariant as the one defined in *UIManual*.

MACHINE NewOIManual
SETS
$CL\_Position = \{sp_1, sp_2, sp_i, stop\}$; (new value sp_i)
VARIABLES
cl_pos
INVARIANT
$cl\_pos \in CL\_Position$
INITIALISATION
$cl\_pos := stop$
OPERATIONS
$Cl\_Speed1 \mathrel{\hat{=}}$ **PRE** $(cl\_pos = sp_2 \vee cl\_pos = stop)$
THEN $cl\_pos := sp_1$ **END**;
. $Cl\_Speed2 \mathrel{\hat{=}}$ **PRE** $cl\_pos = sp_1$
THEN $cl\_pos := sp_2$ **END**;
$Cl\_Stop \mathrel{\hat{=}}$ **PRE** $cl\_pos = sp_1$
THEN $cl\_pos := stop$ **END**

Fig. 4. The new B specification obtained from *OIManual* and the invariant of *UIManual*

The verification with the B theorem prover shows that the new B specification is valid, which means that the operations preserves the invariant. Therefore the operations of the offered interface *OIManual* satisfy the invariant provided by the used interface *UIManual*.

4.1.2 Verification of interfaces compatibility at signature, semantics and protocol levels

The verification of the compatibility between the offered interface *OIManual* and the used interface *UIManual* is based on verifying that the component behavior described by the B specification of the offered interface and its protocol simulates the component behavior described by the B specification of the used interface and its protocol.

The steps which compose this verification approach are:

- Construction of the transition systems that capture the component behavior described in the specification of offered and used interfaces: we exploit the protocol specifications and the B specification of the interfaces to construct these transition systems.
- Verification of the interface compatibility by defining and verifying the relation of simulation between the transition systems obtained in the above step.

Construction of the transition systems corresponding to the interface specifications: The protocols specified in the interfaces *OIManual* and *UIManual* are respectively:

- $(Cl\_Speed1.(Cl\_Speed2.Cl\_Speed1)^*.Cl\_Stop)^*$.

- $((Cl\_Speedi.Cl\_Stop)^*.$
  $(Cl\_Speed1.(Cl\_Speed2.Cl\_Speed1)^*.Cl\_Stop)^*.$
  $(Cl\_Speedi.Cl\_Stop)^*)$.

These protocols are specified with a finite regular language which has the alphabet composed of operation names specified in the interfaces. According to the Kleene theorem [18], we can model these protocols with finite transition systems [4].

The transition systems do not express the total behaviors described in the interfaces, because the values of their states are not significant. Indeed, they only present the information at the levels of operation signature and protocols but not at the level of the semantics of the operation (their pre and postconditions). Therefore, we enhance these transition systems by changing the values of their states. We use the B specification of the interfaces in order to decorate the states of the transition systems by a set of atomic propositions that correspond to pre and postconditions of the operations. The approach of constructing the transition systems is presented as follows:

- From the clause Initialization of the B specification of an interface, we decorate the initial states of the transition system that model the protocol by a set of atomic

propositions which are the equality between the variables and their values.

- From the operations of the B specification of an interface, we decorate the other states by a set of atomic propositions which are the equality between the variables and their new values. The values of variables in a state s_i, such that $t \stackrel{\text{def}}{=} s_{i-1} \stackrel{op}{\rightarrow} s_i$ is a transition, are obtained by applying the generalised substitution of the operation *op* on the variables of the state s_{i-1}.

By applying our approach on the previous example, we obtain the transition systems TS_o (see figure 5) and TS_u (see figure 6) that model both behaviors of software components described respectively by the offered interface *OIManual* and the used interface *UIManual*. These transition systems allow to know all the authorized behaviors for the component CLever corresponding to the specification of the interfaces *OIManual* and *UIManual*.

The transition system $TS_o \stackrel{\text{def}}{=} \langle S_{0o}, S_o, \rightarrow_o, L_o, F_o \rangle$ is composed of the following information:

- the initial states: $S_{0o} = \{s_0\}$
- the set of states: $S_o = \{s_0, s_1, s_2\}$
- the transition relation $\rightarrow_o$ expressed by the following set of transitions: $\{s_0 \stackrel{Cl\_Speed1}{\rightarrow} s_1, s_1 \stackrel{Cl\_Speed2}{\rightarrow} s_2, s_2 \stackrel{Cl\_Speed1}{\rightarrow} s_1, s_1 \stackrel{Cl\_Stop}{\rightarrow} s_0\}$
- the labelling function L_o defined by: $L_o(s_0) = \{cl\_pos = stop\}$, $L_o(s_1) = \{cl\_pos = sp_1\}$, $L_o(s_2) = \{cl\_pos = sp_2\}$
- the set of the final states: $F_o = \{s_o\}$.

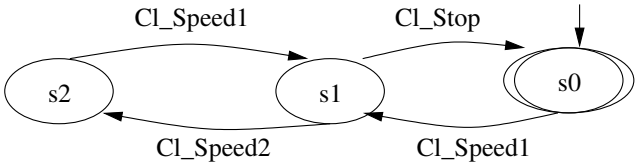

Fig. 5. The transition system TS_o corresponding to the interface *OIManual*

The transition system $TS_u \stackrel{\text{def}}{=} \langle S_{0u}, S_u, \rightarrow_u, L_u, F_u \rangle$ is composed of the following information:

- the initial states: $S_{0u} = \{s'_0\}$
- the set of states: $S_u = \{s'_0, s'_1, s'_2, s'_3\}$
- the transition relation $\rightarrow_u$ expressed by the following set of transitions: $\{s'_0 \stackrel{Cl\_Speed1}{\rightarrow} s'_1, s'_1 \stackrel{Cl\_Speed2}{\rightarrow} s'_2, s'_2 \stackrel{Cl\_Speed1}{\rightarrow} s'_1, s'_1 \stackrel{Cl\_Stop}{\rightarrow} s'_0, s'_0 \stackrel{Cl\_Speedi}{\rightarrow} s'_3, s'_3 \stackrel{Cl\_Stop}{\rightarrow} s'_0\}$
- the labelling function L_u is defined as follows: $L_u(s'_0) = \{cl\_pos = stop\}$, $L_u(s'_1) = \{cl\_pos = sp_1\}$, $L_u(s'_2) = \{cl\_pos = sp_2\}$, $L_u(s'_3) = \{cl\_pos = sp_i\}$
- the set of the final states: $F_u = \{s'_o\}$.

Verification of the relation of simulation: We define a relation of simulation between two transition systems as an adaptation of the definition of the simulation

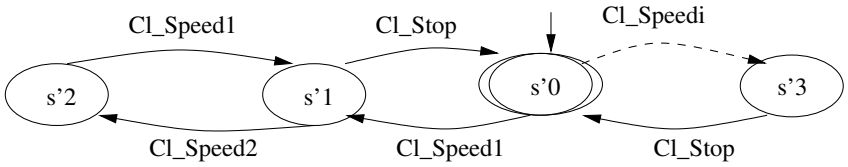

Fig. 6. The transition system TS_u corresponding to the interface *UIManual*

relation of Milner [15]. A transition system TS_0 simulates a transition system TS_1 according to the Milner definition if and only if the system behavior described by TS_1 is included in the system behavior described by TS_0.

In our case, we need to take into account the transition labels in order to verify components compatibility. In Milner's definition the transition are not labelled.

Definition 1 *Let $TS_o = \langle S_{0o}, S_o, \rightarrow_o, L_o, F_o\rangle$ and $TS_u = \langle S_{0u}, S_u, \rightarrow_u, L_u, F_u\rangle$ be two transition systems. Let R be a relation between S_u and S_o, $R \subseteq S_o \times S_u$. R is a relation of simulation iff for each couple of states $(s_i, s'_i) \in S_o \times S_u$ we have:*

- *if $(s_i, s'_i) \in R$, then $L_u(s_i) \subseteq L_o(s'_i)$*
- *if $(s_i, s'_i) \in R$ and $s'_i \stackrel{a}{\rightarrow} s'_{i+1} \in \rightarrow_u$, then there exists $s_{i+1} \in S_o$ such that $s_1 \stackrel{a}{\rightarrow} s_{i+1} \in \rightarrow_o$ and $(s_{i+1}, s'_{i+1}) \in R$.*

After defining the relation of simulation between the states of transition systems, we define what it means that a transition system simulates another transition system.

Definition 2 *Let $TS_o = \langle S_{0o}, S_o, \rightarrow_o, L_o, F_o\rangle$ and $TS_u = \langle S_{0u}, S_u, \rightarrow_u, L_u, F_u\rangle$ be two transition systems. Let R be a relation of simulation between S_u and S_o, $R \subseteq S_o \times S_u$. TS_o simulates TS_u iff $\forall s'_0.(s'_0 \in S_{0u} \Rightarrow \exists s_0.(s_0 \in S_{0o} \ \wedge \ (s_0, s'_0) \in R))$*

According to *Definition 2*, the transition TS_o simulates the transition system TS_u if and only if the simulation relation holds between their initial states.

Theorem 4.1 *Let TS_o and TS_u be two transition systems that model respectively an offered interface OI and a used interface UI. When the relation of simulation R holds between the states of TS_o and TS_u, then the interfaces OI and UI are compatible at signature, semantics and protocol levels.*

Proof. Suppose that TS_o simulates TS_u.

- Compatibility at protocol level: TS_o and TS_u model respectively the interfaces OI and UI. The specification of these interfaces includes the B specification and the protocol specification. According to *Definition 1* and *Definition 2*, the verification of the simulation relation is based on the parallel exploration of the transition systems TS_o and TS_u. Thus we verify that the traces of the paths explored in TS_u are contained in the traces of the paths explored in TS_o. Consequently, verifying the relation of simulation implies verifying whether the set of the operation calls described in UI are included in the set of the operation calls described in OI.

- Compatibility at signature and semantics levels: the operations described in OI and UI are expressed in TS_o and TS_u by transitions. Source and target states of these transitions are decorated with a set of atomic propositions which expresses respectively pre and postconditions of the operations. Furthermore transitions are labelled with operation names. According to *Definition 1* and *Definition 2*, the verification of the simulation relation implies the verification of the matching of transitions between TS_o and TS_u at signature and semantics (operation names, pre and postconditions) levels. This involves the verification of the compatibility at signature and semantics levels.

□

Case study: In order to verify that the interface *OIManual* is compatible with the interface *UIManual*, we must verify that the relation of simulation R holds between the two transition systems TS_o and TS_u. The verification requires the parallel exploration of the transition systems TS_o and TS_u by beginning from their initial states. Therefore, the proof that TS_o simulates TS_u is the proof that the initial states $(s_0, s'_0) \in R$ such that s_0 is the initial state of TS_o and s'_0 is the initial state of TS_u.

We illustrate the verification algorithm of the simulation relation by verifying whether TS_o simulates TS_u as follows:

- verify the first condition of the relation on the couple (s_0, s'_0): the condition is verified because $L_o(s_0) = L_u(s'_0)$,
- verify the second condition on (s_0, s'_0): there is a transition $s'_0 \overset{Cl\_Speedi}{\rightarrow} s'_3 \in \rightarrow_u$ in TS_u, but there is not a transition $s_0 \overset{Cl\_Speedi}{\rightarrow} s_3 \in \rightarrow_u$ such that $(s_3, s'_3) \in R$. Therefore $(s_0, s'_0) \notin R$.

The verification of the simulation relation between the transition systems TS_o and TS_u shows that TS_o does not simulate TS_u. That means that the interfaces *OIManual* and *UIManual* are not compatible. Therefore, the behavior required by the component WWiper is not offered by the component CLever. The verification fails because the component WWiper requires the operation *Cl_Speedi* that is not offered by the component CLever (see the dashed transition in the transition system TS_u in figure 6).

4.2 Verification of component substitutability

To determine whether a defective component can be substituted by another component, we propose an approach based on the verification of the relation of *equivalence* between component specifications. This approach allows to determine whether the behavior of the defective component is equivalent to the behavior of the substitute component.

In order to illustrate this approach, we present in the figure 7 two components control lever *CLever* and *CLever'*. Suppose that we need to replace the component *CLever* by the component *CLever'*. To reach this goal, the component *CLever'* must satisfy the following constraints:

- The set of services required by the component *CLever'* must be the same as the

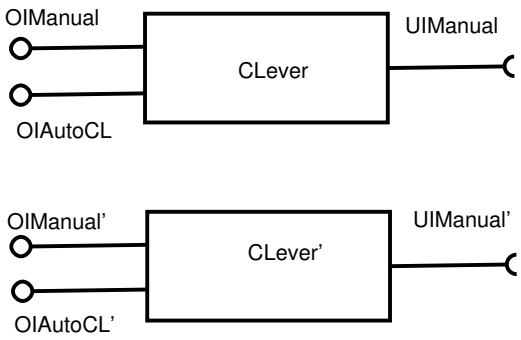

Fig. 7. component substitutability

set of the services required by the component *CLever*. To check this constraint it is necessary to verify the equivalence between component behaviors described in the used interfaces of *CLever* and *CLever'*. In our example (see figure 7), we verify that the behavior described in the interface *UClever* is equivalent to the one described in the interface *UClever'*.

- The set of the services offered by the component *CLever'* must be the same as the set of the services offered by the component *CLever*. To check this constraint it is necessary to verify the equivalence between component behaviors described in the offered interfaces. In our example, we verify that the behavior described in the interface *OIManual* is equivalent to the one described in the interface *OIManual'*, and we also verify that the behavior described in the interface *OIAutoCL* is equivalent to the one described in the interface *OIAutoCL'* .

To verify the equivalence between the component behaviors, we propose an approach based on the B method and on a relation between transition systems. This approach is divided into two steps:

- **Equivalence between invariants**: we verify the equivalence between the invariants defined in the offered interfaces and between the invariants defined in the used interfaces. In our example, we verify that the invariant defined in the offered interface *OIManual* is equivalent to the invariant defined in the interface *OIManual'*. We use the B theorem prover and the B specification of the interfaces to perform this verification.
- **Equivalence between component interfaces**: in this step, we verify the equivalence between component behaviors described in the offered interfaces and the equivalence between component behaviors described in the used interfaces. In the example in figure 7, we verify equivalence between component behaviors described in *OIManual* and *OIManual'*, and between *OIAutoCL* and *OIAutoCL'*. We also verify equivalence between component behaviors described in *UClever* and *UClever'*. To perform this verification we define and we check the relation of *bisimulation* (this relation is defined in the next paragraph) between the transition systems that model the component interfaces (relation of bisimulation between offered interfaces and between used interfaces). So, we verify that the relation of bisimulation holds between the following transition systems
  · the transition systems describing the offered interfaces *OIManual* and *OIMan-*

ual',

- the transition systems describing the offered interfaces *OIAutoCL* and *OIAutoCL'*,
- the transition systems describing the used interfaces *UClever* and *UClever'*.

In the following, we define the relation of bisimulation between transition systems. This relation is based on the relation of simulation defined in the last section.

A transition system TS' bisimulates a transition system TS if and only if the system behavior described by TS' is the same as (we say also that is equivalent) to the system behavior described by TS.

Definition 3 *Let* $TS = \langle S_0, S, \rightarrow, L, F\rangle$ *and* $TS' = \langle {S_0}', S', \rightarrow', L', F'\rangle$ *be two transition systems. Let* R *be a relation of simulation between* S *and* S'*. Let* α *be another relation between* S *and* S'*,* $\alpha \subseteq S \times S'$*.* α *is a relation of bisimulation iff for each couple of states* $(s_i, s'_i) \in S \times S'$ *we have:*

if $(s_i, s'_i) \in \alpha$*, then* $(s_i, s'_i) \in R$ *and* $(s'_i, s_i) \in R$

After defining the relation of bisimulation between the states of transition systems, we define what means that a transition system bisimulates another transition system.

Definition 4 *Let* $TS = \langle S_0, S, \rightarrow, L, F\rangle$ *and* $TS' = \langle {S_0}', S', \rightarrow', L', F'\rangle$ *be two transition systems. Let* α *be a relation of bisimulation between* S *and* S'*,* $\alpha \subseteq S \times S'$*.* TS' *bisimulates* TS *iff* $\forall s_0.(s_0 \in S_0 \Rightarrow \exists s'_0.(s'_0 \in S'_0 \ \wedge \ (s_0, s'_0) \in \alpha))$

According to *Definition 4*, the transition TS' bisimulates the transition system TS if and only if the bisimulation relation holds between their initial states.

5 Related Work

Several works have been proposed in the context of specifying component interfaces and verifying their compatibility.

Cheesman and Daniels [7] propose to specify component interfaces using UML [17] and OCL [23] notations. The interface specification includes: an information model that provides the type of the information expressed in the interfaces (it is specified using UML class diagram), invariants on the information model, and operation specifications that express operation signatures and pre- and postconditions. However this work does not propose any verification approach of interface compatibility.

Yellin and Storm [25] propose a state machine based approach to specify protocols. Protocols are expressed in terms of abstract states and transitions. We are inspired by this work in order to model protocols. In contrast with our proposition, the approach of Yellin and Storm does not cope with the component interoperability at the semantics level but only at the protocol level.

Canal and al [6] use a subset of the polyadic *pi*-calculus to deal with component interoperability only at the protocol level. The *pi*-calculus is very well suited for describing component interactions. The limitation of this approach is the low-level description of the used language and its minimalistic semantics which does not

provide rich feedback to system designers when errors are detected during protocol checks.

In [5], Bastide et al use Petri nets to specify the behavior of CORBA objects, including operation semantics and protocols. This work correlates with our approach because we use transition systems that can be expressed as Petri nets, to express component behaviors. The difference to our approach is that we take into account the invariant in the interface specifications and we use the B approach to verify interface compatibility.

In [10], J. Han specifies protocols with a temporal logic based approach. This approach leads to a rich specification for component interfaces.

Alfaro and Henzinger [3] propose an interesting approach which allows the verification of the interfaces compatibility based on the automata and game theories. This approach is well suited for checking the interface compatibility at the protocol level.

In [26], Zaremski and Wing propose an interesting approach to compare two software components. It determines whether one component can be substituted for another. They use formal specifications to model the behavior of components and exploit the Larch prover to verify the specification matching of components.

The approaches described above treat the component interoperability at the semantics and the signature levels, or at the protocol level, not both. However, in this paper we propose an approach which handles the component interoperability and component substitutability at the both levels.

An interesting approach proposed in [19], allows to formally specify, refine and verify component based systems. It is based on Object-Z language and the process algebra CSP. The main difference with our approach is the specification language. In our case, we have chosen the B method to specify component interfaces in order to exploit the B theorem prover.

6 Conclusion

This paper presents an approach to verify component interoperability. We have considered two cases in the interoperability: component compatibility and component substitutability. This approach is based on the use of the B method for specifying component interfaces. An interface specification provides the following information:

- an initialization predicate which provides initial states of a component
- a list of operations, specified by means of their preconditions and generalized substitutions
- an invariant property that must be respected by the initialization predicate and the operations.

Specification of interfaces is enhanced by the specification of protocols with finite transition systems.

The approach to verify the component compatibility is composed of two steps. First, we verify with the B theorem prover that the invariant specified in a used interface is satisfied by the specification of the appropriate offered interface. Sec-

ond, we use protocol specifications and B specification of interfaces to complete the verification of interface compatibility at the signature, semantics, and protocol levels by verifying the simulation relation between the transition systems that model component behaviors.

The approach to verify the component substitutability is based on the B method and the verification of the relation of bisimulation between transition systems that models offered interfaces and between transition systems that models used interfaces (interfaces of the defective component and the substitute component).

References

[1] J. Hernandez A. Vallacillo and M. Troya. Object interoperability. In *Object Oriented Technology: ECOOP'99 Workshop Reader*, pages 1–21, 1999.

[2] J.-R. Abrial. *The B Book*. Cambridge University Press - ISBN 0521-496195, 1996.

[3] L. Alfaro and T. A. Henzinger. Interface automata. In *9 th Annual Aymposium on Foundations of Software Engineering, FSE*, pages 109–120. ACM Press, 2001.

[4] A. Arnold. Mec: a system for constructing and analysis transition systems. In *AMAST*, pages 81–82, 1999.

[5] R. Bastide, O. Sy, and P. A. Palanque. Formal specification and prototyping of CORBA systems. In *ECOOP '99: Proceedings of the 13th European Conference on Object-Oriented Programming*, pages 474–494. Springer-Verlag, 1999.

[6] C. Canal, L. Fuentes, E. Pimentel, J-M. Troya, and A. Vallecillo. Extending CORBA interfaces with protocols. *Comput. J.*, 44(5):448–462, 2001.

[7] J. Cheesman and J. Daniels. *UML Components – A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2001.

[8] L. Doldi. *UML 2 Illustrated - Developing Real-Time & Communications Systems*. TMSO, 2003.

[9] J. Han. A comprehensive interface definition framework for software components. In *The 1998 Asia Pacific software engineering conference*, pages 110–117. IEEE Computer Society, 1998.

[10] J. Han. Temporal logic based specification of component interaction protocols. In *Proceedings of the Second Workshop on Object Interoperability ECOOP'2000*, pages 12–16. Springer-Verlag, 2000.

[11] D. Konstantas. Interoperation of object oriented application. In Oscar Nierstrasz and Dennis Tsichritzis, editors, *Object-Oriented Software Composition*, pages 69–95. Prentice Hall, 1995.

[12] O. Kouchnarenko and A. Lanoix. Refinement and verification of synchronized component-based systems. In *Proc. Int. Sympo. of Formal Methods (FME 2003)*, pages 341–358, Pisa, Italy, 2003.

[13] B. Meyer. *Object-Oriented Software Construction*. Prentice-Hall, second edition, 1997.

[14] Microsoft Corporation. *The Component Object Model Specification, Version 0.9*, 1995. http://www.microsoft.com/com/resources/comdocs.asp.

[15] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[16] The Object Mangagement Group (OMG). *The Common Object Request Broker: Architecture and Specification, Revision 2.2*, February 1998. http://cgi.omg.org/library/corbaiiop.html.

[17] J. Rumbaugh, I. Jacobsen, and G. Booch. *Unified Modeling Language Reference Manual*. Addison-Wesley, 1997.

[18] S.-C.Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, Princeton, New Jersey, USA, 1956.

[19] G. Smith and J. Derrick. Specification, refinement and verification of concurrent systems - an integration of Object-Z and CSP. *Formal Methods in Systems Design*, 18:249–284, May 2001.

[20] STERIA. *Atelier B : Preuves et Exemples*.

[21] Sun Microsystems. *JavaBeans Specification, Version 1.01*, 1997. http://java.sun.com/products/javabeans/docs/spec.html.

[22] C. Szyperski. *Component Software*. ACM Press, Addison-Wesley, 1999.

[23] J. Warmer and A. G. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1999.

[24] P. Wegner. Interoperability. *ACM Computing Survey*, 28(1):285–287, 1996.

[25] D. M. Yellin and R. E. Strom. Protocol specifications and component adaptors. *ACM Trans. Program. Lang. Syst.*, 19(2):292–333, 1997.

[26] Amy Moormann Zaremski and Jeannette M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.