D3.4

Version: 1
Date: 2011-02-04
Dissemination status: PU
Document reference: D3.4

Management tools for compliant behaviour

Project acronym:	COMPAS
Project name:	Compliance-driven Models, Languages, and Architectures for Services
Call and Contract:	FP7-ICT-2007-1
Grant agreement no.:	215175
Project Duration:	01.02.2008 – 28.02.2011 (36 months)
Co-ordinator:	TUV Technische Universitaet Wien (AT)
Partners:	CWI Stichting Centrum voor Wiskunde en Informatica (NL)
	UCBL Université Claude Bernard Lyon 1 (FR)
	USTUTT Universitaet Stuttgart (DE)
	TILBURG UNIVERSITY Stichting Katholieke Universiteit Brabant (NL)
	UNITN Universita degli Studi di Trento (IT)
	TARC-PL Telcordia Poland (PL)
	THALES Thales Services SAS (FR)
	PWC Pricewaterhousecoopers Accountants N.V. (NL)

This project is supported by funding from the Information Society Technologies Programme under the 7th Research Framework Programme of the European Union.

Project no. 215175

COMPAS

Compliance-driven Models, Languages, and Architectures for Services

Specific Targeted Research Project

Information Society Technologies

Start date of project: 2008-02-01 Duration: 36 months

D3.4 Management tools for compliant behaviour

Revision 0.4

Due date of deliverable: 2010-12-31

Actual submission date: 2011-02-04

Organisation name of lead partner for this deliverable:

UCBL – Université Claude Bernard Lyon 1, France

Contributing partner(s):

CWI – Stichting Centrum voor Wiskunde en Informatica, Netherlands

Project funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

History chart

Issue	Date	Changed page(s)	Cause of change	Implemented by
0.1	2010-12-23	All sections	New document	UCBL
1.0	2010-12-31	Document	Approve & Release	TUV

Authorisation

No.	Action	Company/Name	Date
1	Prepared	UCBL	2010-12-12
2	Approved	TUV	2010-12-31
3	Released	TUV	2010-12-31

Contents

List of figures

Abstract

This deliverable describes the adaptation of compatibility and replaceability tools of deliverable [D3.3] to the COMPAS infrastructure.

Compatibility refers to the fact that two services can interact without problem, which means that the behaviour of each of them fulfils the other's expectations. This is of particular importance when the compliance of the service with some compliance request relies on the others' compliance. Moreover, would one of the services need to be replaced, it should be replaced with one that fulfils the other's expectations without having to check the whole process again. This approach was demonstrated in deliverable [D3.3] on credential based access control policies. This deliverable describes the reimplementation of this library in a generic way, protocol being parameterized by transition specifications that reflects compliance requirements and compliance fulfilments. We demonstrate how linear temporal logic, which is used as an underlying language for compliance requests in COMPAS [D2.6], can be used as transition specification. We also show how the library allows combining various kinds of specifications, taking care of the behaviour induced interferences.

This deliverable limits its objectives in providing information on the use of the generic compatibility and replaceability library, some implementation details and location of sources of the compatibility library.

1. Introduction

Compliance is a term generally used to refer to the conformance to a set of laws, regulations, contracts, or best practices (compliance sources according to COMPAS conceptual model [D7.1]).

Model assessment (also referred to as model compliance) addresses the issue of conformance towards a policy, standard, law or technical specification that has been clearly defined. This also includes but is not limited to conform towards business regulations and stated user service requirements [D7.1]. As the authors in [EY09] point out in their survey, compliance toward regulations is finding more and more attention in the eyes of the research community. Applications of model assessment include workflow checking, protocol verification, and constraint validation [MMZ05]. Factors that motivate model compliance utility are: cost of the implementation prototype just for assessing the model, cost of re-implementing once that assessment results are negative, risk of testing on real-world already deployed systems, complexity of designed systems which prohibits exhaustive static verification and validation.

In order to lighten the cost reimplementation in case of negative assessment results and to ease the integration of legacy components, one should be able to check compliance requirement not only at the level of the whole system, but also at the level of each component. Behaviour of components can be represented as compliance annotated business protocols. Compatibility notion allows checking whether a component's behaviour complies the one that is expected by the other components, in particular in terms of the compliance annotation of the protocol. On another hand, replaceability is the dual notion stating that a component can replace another one without violating other components' expectations. If a component cannot be used it is also important to indicate which parts of the components behaviour diverges from the expected one and how it could affect the compliance of the overall process. A first

implementation of compatibility and replaceability checking has been demonstrated in [D3.3]. This deliverable describes the reimplementation of this library in a generic way, in order to support various compliance annotations on the behaviour of components as well as the combination of such compliances annotations.

1.1. Purpose and scope

Error! Reference source not found. provides an illustration of the relationship between the compatibility tools library the other components of the COMPAS architecture. The compatibility tools library is part of the process verification tools and the annotated business protocols will be stored in the Fragmento service, which implements the process repository (described in [D4.4]). Whenever a component is to be used the currently designed process, the compatibility library can be used to check if the component can be used in the process. Business protocols an also be obtained by transforming a constraint automaton describing the behaviour of a Reo process [D3.2]. The generic implementation of the library allows for using the compliance request language [D2.2, D2.6] as a compliance annotation for a business protocol.

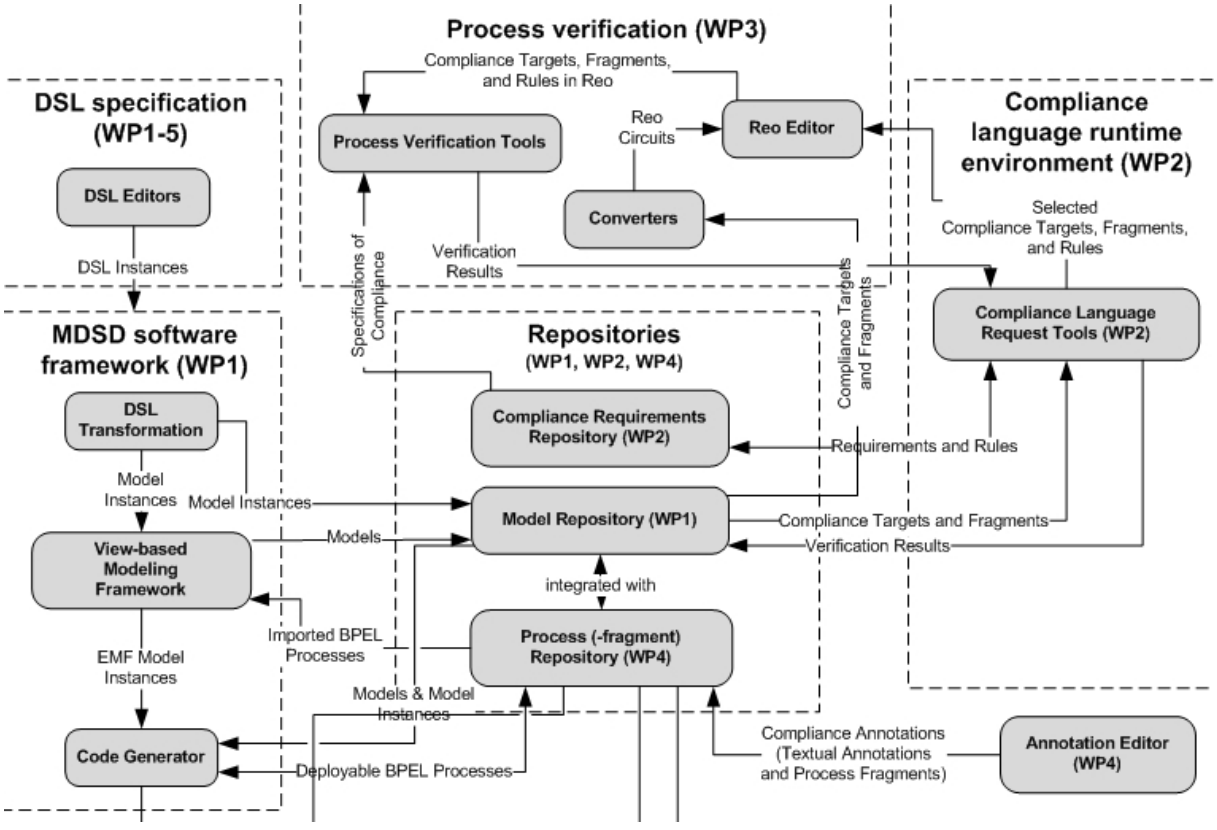

Figure 1 Part of the global COMPAS architecture

1.2. Document overview

The deliverable is organized as follows: Section 2 provides a user guide for the compatibility library. Section 3 gives an overview on the implementation details of the library, especially how annotations can are combined. Section 4 provides information on where the reader can download the source code of the library. Finally, Section 5 draws conclusions.

2. User guide for the compatibility library

2.1. Generic interfaces for transition specifications

Transition specifications

The library relies on a set of interfaces abstracting on one side the protocol implementation details and on the other side the transition specification. Transition specifications are instances of data domains described in [D3.1]. The Java interface for transition specifications is parameterized by the type of transitions specifications. In most cases this parameter is instantiated by the class of the implementation or the root class if the implementation of the interface relies on a hierarchy of classes. There can be three kinds of transitions specifications: for incoming messages, for outgoing messages and products of an incoming specification with an outgoing one. The distinction between the three kinds is only necessary when the specification is asymmetric between incoming and outgoing messages, like for access control policies and credentials and as opposed *e.g.* to schema constraints on messages or message meaning expressed in description logics. Some specifications can also be cumulated, which means that the actual value of the specification is obtained by adding to the current specification the specifications appearing on the different paths in the protocol leading to the current transition (e.g. accumulation of access control credentials).

We describe now the main methods of the `TransitionSpecification` interface. In the following, the type E is the parameter, which will be instantiated when implementing the interface.

- `E and(E spec)`: should return the specification expressing that both specifications, *this* and spec are fulfilled.
- `E product(E spec, boolean direction)`: should return a product transition specification matching `this` and `spec`. The direction parameter is true if `this` is considered as an incoming specification and spec as an outgoing one. Note that if the specification is symmetrical then `product` and `and` methods should return the same result.
- `boolean in(Collection<E> specs)`: states whether `this` is included in the collection of specifications. From a logical point of view, the inclusion is to be checked with reference to the disjunction `specs`, that is `this` can span over several elements of `specs`.
- `E cumulate(E prevState, boolean isIncomming)`: should return the result of adding to `this` a specification corresponding to the specifications encountered on the different paths leading to the state from which the current transition starts. The accumulation only occurs on incoming or outgoing transition, not on product transitions, therefore `isIcomming` can be used to determine the kind of transition specification.

Combination of transition specifications

The library provides a particular implementation (`CombinedTransitionSpecification<E extends TransitionSpecification<E>, F extends TransitionSpecification<F>>`) of the transition specification interface that allows one to combine two implementations, provided that they do not directly interfere one with the other. For example access control and

message meaning specification can be considered as non interfering unless the message meaning specification allows to say something about access control and credentials.

Time related issues

Implicit timeout transitions are transformed into additional time specifications on other explicit transitions (see [D3.3]). For handling this transformation, the `TimeSpecificationContainer` interface must be implemented. This interface extends the `TransitionSpecification` interface and references `TimeSpecification` for handling time related issues. The `TimeSpecification` class implements the `TransitionSpecification` and represents simple time constraints using time intervals.

We describe now the main additional methods for `TimeSpecificationContainer`.

- `TimeSpecification getTimeSpecification()`: should return the timed part of the specification.
- `E shift(double shiftValue)`: should return a specification of the same type (E) that is equivalent to `this` except for the timed part, which should be shifted by given amount.
- `E timedAnd(TimeSpecification ts)`: should return an equivalent specification except for the timed part, which is to be combined with the provided time specification.

The `CombinedTransitionSpecification` has been extended to support timed aspects. More precisely the second specification in `TimeCombinedTransitionSpecification` is expected to be a `TimeSpecificationContainer` and al time related operations are delegated to it. Note that only one of the two specifications can be time related as otherwise it means that they could interfere one with the other.

2.2. Implementing a new kind of specification

In this section we give an example of implementation transition specification based on linear temporal logic, which is the backend language for compliance requests [D2.2, D2.6]. A formula on an incoming transition expresses that the service will fulfil the formula upon reception of the message. A formula on an outgoing transition expresses that the service expects the formula to be fulfilled when the message is sent.

We assume that LTL formulas are represented through a hierarchy of classes (named LTLXY, where XY is the name of the LTL operator) and that the following methods are available on those classes: `isTrue()` and `step()`. The first method can be implemented by converting the formula to a Büchi automaton. The step method returns a LTL formula representing what should be fulfilled after one step on the Kripke structure.

Since the requirement is put on incoming messages, the intuitive interpretation has to be reversed: the formula on the incoming transition has to be stronger than the one on the outgoing transition. This will impact the `in()` method.

Implementation of and and product

The implementation of those methods are straightforward, using the LTLAnd(,) constructor:

```
public AbstractLTLFormula product(AbstractLTLFormula spec, boolean dir) {
      return new LTLAnd(this,spec);
}
```

Implementation of in()

This method must check whether this is covered by the collection of LTL formulas. Since this represents the expected behaviour, it should imply the disjunction of the provided formulas:

```
public boolean in(Collection<AbstractLTLFormula> specs) {
      AbstractLTLFormula f = null;
      boolean first = true;
      for(AbstractLTLFormula f2 : specs) {
            if (first) {
                  f = f2;
                  first = false;
            } else {
                  f = new LTLOr(f,f2);
            }
      }
      return (new LTLOr(new LTLNot(this),f)).isTrue();
}
```

Implementation of cumulate()

This method is used to cumulate previous assertions. Its implementation mainly relies on the `step()` method.

```
public AbstractLTLFormula cumulate(AbstractLTLFormula prevState,
            boolean isIncomming) {
      // cumulate on incomming transitions only
      if (isIncomming) {
            return new LTLAnd(prevState.step(),this);
      } else {
            return this;
      }
}
```

2.3. Business protocols

The business protocols are represented using the ProtocolImpl class. The class provides methods for adding transitions, checking compatibility and replaceability. It also contains a method for transforming implicit transition into time specifications, which is necessary for compatibility checking. The class is parameterized by the class of states and the class of transition specifications. The transition specifications should be time aware. A time aware transition specification can be obtained by combining with a `TimeSpecification` using a `TimeCombinedTransitionSpecification`.

3. Implementation details

The library is build as a maven project, which eases the integration into other Java code.

Compatibility and replaceability checking

Compatibility checking involves four steps that delegate all transition specification related work to the transition specification implementations.

The first step is implicit transition elimination. It translates transition occurring after an implicit transition to the start state of the implicit transition, adding a time constraint to represent the effect of the implicit transition. It also adds time constraint to transition starting

in the same state as an implicit transition to represent the fact that the transition cannot occur after the implicit transition has been taken.

The second step consists in cumulating specifications that need to be cumulated (such as credentials for access control specifications and formulas in incoming transition for LTL). The algorithm works by first assigning specifications to states representing the effect on the transitions ending in this state and then updating those specifications by including the effect of the specification of the starting state of a transition into the transition's specification. This update step is repeated until a fixpoint is reached. The transitions specifications are then updated using the specification of the start state of the transition.

The third step consists in checking that each outgoing transition is fully matched by a set of transitions in the product automata. This is where the method `in()` is used. The algorithm proceeds by simply traversing the product automaton.

The fourth step consists in checking whether each reachable pair of states in the product automaton is the starting point of a path to a final state.

Replaceability checking works in a similar way.

Specification combination

Combining two independent specification classes is straightforward, except for the `in()` method, by delegating to both specifications. The `in()` method requires special care as illustrated by the following example. Consider a first incoming transition with time interval specification [1,10] and a LTL specification X (a or c) (next a or c is true), and a second specification with [9,20] as time specification and X (b or c) as LTL specification. Their "specification by specification union" would be [1,20] and X (a or b or c). But this includes [1,8] and X b which is neither covered by the first or the second specification. However [5,15] and X c is covered by their combination. The implementation of the `in()` method proceeds by transforming the comparison into an equivalent set of comparisons that requires only comparison between specifications of the same kind (*e.g.* only time with time and LTL with LTL). The cost of the transformation can be exponential in the size of the specification collection to compare to. We plan to apply frequent itemset datamining techniques in order to tackle the potential cost of this comparison.

4. Source Code and additional information

The compatibility tools code is released as an open source project under the GNU General Public License (GPLv3).

The sources are available at https://svn.liris.cnrs.fr/ecoquery/protocols/CompatibilityTools. Compilation requires JDK version 1.6 and the maven 2 (http://maven.apache.org/) build tool.

5. Conclusion

This deliverable presented a generic library implementing compatibility and replaceability checks. It also presented how new kinds of transition specifications can be used to parameterize the library. Further evolutions of the tool will include several additional transition specifications to be provided with the library as well as service based access to the library.

6. Reference documents

6.1. Internal documents

[D2.2] "Initial Specification of Compliance Language Constructs and Operators", ver. 2.0 of 2009-06-08.

[D2.6] "Implementation of an Integrated Prototype handling Interactive User Specified Compliance Requests in a Compliance Language", ver. 1.1 of 2009-12-17.

[D3.1] "Specification of a Behavioral Model for Services", ver. 2.0 of 2009-06-08.

[D3.2] "Visual Environment for Service Description", ver. 1.4 of 2009-07-31.

[D3.3] "Compatibility_and_replaceability_tools", ver. 1.0 of 2009-12-22.

[D4.4] "Supporting infrastructure", ver. 1.0 of 2009-12-22.

[D7.1] "Public Web-Site", http://www.compas-ict.eu

[DoW] "Description of Work", ver. 15 of 2007-09-25.

6.2. External documents

[EY09] E.Young. European fraud survey 2009. Available at: http://www2.eycom.ch/publications/items/fraud_eu_2009/200904_EY_European_Fraud_Survey.pdf

[MMZ05] E. D. Maria, A. Montanari, M. Zantoni. Checking workflow schemas with time constraints using timed automata. Proceedings of OTM Workshops, 2005.