

D2.5

Version: 1.1

Date: 2010-07-31

Dissemination status: PU

Document reference: D2.5



Specification of an Extension to BPEL for Adaptability – Final Version

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 Università 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

**D2.5 Specification of an Extension to BPEL
for Adaptability – Final Version**

Revision 1.1

Due date of deliverable: 2010-07-31

Actual submission date: 2010-07-31

Organisation name of lead partner for this deliverable:

USTUTT Universitaet Stuttgart, DE

Contributing partners:

TILBURG UNIVERSITY Stichting Katholieke Universiteit Brabant, NL

CWI Stichting Centrum voor Wiskunde en Informatica, NL

TUV Technische Universitaet Wien, AT

UNITN University of Trento, IT

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	2010-03-15	All sections	New document	USTUTT
0.2	2010-04-07	Introduction	Add deliverable positioning	USTUTT
0.3	2010-05-03	All sections	Prepared deliverable draft	USTUTT, TILBURG UNIVERSITY, CWI
0.4	2010-05-27	Methodology	Added content	USTUTT, TILBURG UNIVERSITY, CWI
0.5	2010-06-15	All sections	Added content	USTUTT
0.6	2010-07-04	All sections	Refinement and revision	TILBURG UNIVERSITY
0.7	2010-07-08	Section 3.6	Added content	CWI
0.8	2010-07-12	Sections 3.1, 3.4, 3.5	Added and revised content	USTUTT
0.9	2010-07-15	Section 4	Added content	USTUTT, TILBURG UNIVERSITY
1.0	2010-07-16	Section 3	Internal review	TILBURG UNIVERSITY, USTUTT, CWI, UNITN, TUV
1.1	2010-07-31		Approve & Release	TUV

Authorisation

No.	Action	Company/Name	Date
1	Prepared	USTUTT	2010-03-15
2	Approved	TUV	2010-07-31
3	Released	TUV	2010-07-31

Disclaimer: The information in this document is subject to change without notice. Company or product names mentioned in this document may be trademarks or registered trademarks of their respective companies.

All rights reserved.

The document is proprietary of the COMPAS consortium members. No copying or distributing, in any form or by any means, is allowed without the prior written agreement of the owner of the property rights.

This document reflects only the authors' view. The European Community is not liable for any use that may be made of the information contained herein.

Contents

1. Introduction	6
1.1. Purpose and scope	6
1.2. Document overview	8
1.3. Definitions and glossary	8
1.4. Abbreviations and acronyms	9
2. Integration into the overall COMPAS architecture	9
3. Methodology for assurance of compliant business process design	13
3.1. Overall proceeding	13
3.2. Definition of compliance requirements	14
3.3. Formalization of compliance requirements	15
3.4. Search and creation of a suitable compliance fragment	20
3.5. Integration of a compliance fragment into a process	23
3.6. Verification of the process	26
4. Conclusion	31
4.1. Summary	31
4.2. Limitations of the approach	32
4.3. Outlook	32
Reference documents	32
4.4. Internal documents	32
4.5. External documents	33

List of figures

Figure 1	COMPAS conceptual model	10
Figure 2	Overall COMPAS architecture	12
Figure 3	Overall proceeding for assurance of compliant business process design	13
Figure 4	Definition of compliance requirements	14
Figure 5	Defining compliance requirements and sources in CRLT	15
Figure 6	Formalization of compliance requirements	15
Figure 7	Conceptual model for pattern expressions within CRL	16
Figure 8	Rule Modeller	18
Figure 9	Creating compliance requests	19
Figure 10	A user interface with compliance check results displayed to the user	20
Figure 11	Search and creation of a suitable compliance fragment	20
Figure 12	Search in Fragmento	21

Figure 13	Abstract compliance fragment for an approval [STK+10].....	23
Figure 14	Integration of a compliance fragment into the process	23
Figure 15	Abstract compliance fragment for approval and its concretization for a concrete usage scenario [STK+10]	25
Figure 16	Verification of the process	26
Figure 17	Integration with the Process Verification Tools.....	27
Figure 18	Request management tools: repository	27
Figure 19	Request management tools: verification request details	28
Figure 20	Process formalization: compliance fragment is converted to Reo [STK+10]....	28
Figure 21	Formal process model refinement: data constraints.....	29
Figure 22	Request management tools: verification results.....	30

List of tables

Table 1	Example compliance requirements from the loan origination scenario.....	17
Table 2	Structure of the verification request.....	27
Table 3	Structure of the verification results.....	31

List of listings

Listing 1	Conceptual algorithm for compliance fragment integration.....	25
-----------	---	----

Abstract

This deliverable describes the integration of the results achieved in the field of compliance languages (WP2), process reuse concepts (WP4), and concepts for formal verification (WP3). In particular, we present a methodology to assure compliant business process design. The methodology combines the advantages of logical formulae and verification with reusable process structures related to compliance. Logical formulae and verification provide proof of a compliant design and reusable process structures enable a fast and consistent augmentation of a process with compliance. The solution specified in this deliverable makes both approaches more complete: On the one hand, verification of formal compliance requirements gives proof about whether the process design is compliant, but it does not provide assistance for augmentation with compliance if required. On the other hand, process fragments for compliance enable a fast and consistent integration of compliance into a process, but they do not cover if the integration has been performed in a correct manner, i.e., if the process augmented with compliance is actually compliant by design. Combined, we can exploit the strength of each approach without its shortcoming.

1. Introduction

1.1. Purpose and scope

In former works of WP2 we have been working on expressive languages for compliance concerns. In [D2.1] we provided an overview of the state-of-the-art in compliance languages, particularly focusing on languages for regulatory and legislative provisions. Subsequently, in [D2.2] we introduced an initial version of the Compliance Request Language (CRL), which can be used for the formal specification of compliance requirements that stem from legislative and regulatory bodies, e.g. Sarbanes-Oxley Act (SOX) and Basel II. In [D2.3] we presented the design of a software environment for CRL (the implementation is documented in [D2.6]). This environment enables a user to specify compliance requirements at different levels of abstraction, and it can be used to issue compliance requests in order to identify if (and how) a process can or should be changed to make it compliant. In addition, [D2.3] also contains the meta-model for CRL and proposes extensions to address limitations identified in [D2.2]. In [D2.4] we present an integral part of the CRLT - Rule Modeller - that allows compliance experts to visualize compliance requirements using patterns and generate formal compliance rules based on these visual expressions.

In former works of WP3 we developed a formal behavioural model which we discussed in [D3.1]. Based on this model we presented visual tool prototypes for service specification and verification [D3.2]. The CRL environment integrates with the concepts and tools developed in WP3. For this integration the prototypes provided in [D3.2] are important, as they enable that compliance requests issued by a user can be evaluated using advanced methods for process verification to assure compliance.

In former works of WP4 we have been working on reusable process structures which are related to compliance. In [D4.1] we provided a state-of-the-art report on existing approaches to improving reusability of processes and service compositions. In this deliverable we sketched a reuse concept for usage in the field of compliance, which is called process fragment. In [D4.2] we have elaborated on this concept and defined extensions to the Business Process Execution Language (BPEL) in order to provide support for process fragments for compliance (compliance fragments for short). The approach of compliance fragments aims at ensuring a fast and consistent specification and integration of compliance

structures. A compliance fragment implements a compliance requirement by means of activities and control dependencies between them. We distinguish between two types of compliance fragments:

- i. Annotation Business Process Fragment can be used as an annotation to constraint how a process “should” behave. They, however, do not add or explicitly change behaviour of a process. Annotation Business Process Fragments are evaluated during monitoring either using tools for Business Protocol Monitoring or Complex Event Processing (CEP) based Compliance Monitoring, both developed in WP5 (see [D5.4] for further details). This type of compliance fragment is out of scope of this deliverable.
- ii. Business Process Logic Fragments (BP Logic Fragment) aim at the augmentation of a business process with additional behaviour. When a BP Logic Fragment is integrated into a process, the process fragment is customized to the specific needs of the context in which it is used (i.e. concretized). During integration the extension constructs are removed which results in a regular process, though augmented with compliance. This augmented process can be executed by a standard process engine as we discussed in [D4.4]. Compliance fragments are stored in a process fragment repository, which we initially presented in [D4.4] and recently also shared with the research community [SKL+10]. In this deliverable we focus on this type of compliance fragments.

The description of work [DoW] primarily states that this deliverable (D2.5 “Specification of an Extension to BPEL for Adaptability – Final Version”) integrates the results from D2.1-D2.4 with the process fragment concepts for BPEL developed in WP4 and that it specifies the solution. In [D4.2] we already proposed extensions to BPEL to support compliance fragments which increase adaptability of processes during design time. These extensions include adding Universally Unique Identifiers (UUIDs) to process constructs, which account for traceability. The preparation of a standardization proposal regarding these extensions is ongoing work in WP4. The extensions elaborated in the standardization proposal are already sufficient (and suitable) to integrate the compliance fragments concept from WP4 with the compliance language concepts developed in WP2 in a loosely coupled and flexible manner.

Therefore, in this deliverable we show how the advantages of compliance checking based on logical formulae (WP2, WP3) can be combined with our approach for process structure reuse (WP4). The solution we propose in this deliverable builds on the extensions to BPEL we have already defined in [D4.2]. The essential concepts presented in this deliverable have already been peer-reviewed and accepted for publication [STK+10]. In principle, the solution we propose is not limited to be used with BPEL, but is basically also applicable to other graph-based process languages such as the Business Process Modeling Notation (BPMN) [OMG10].

Regarding the scope of this deliverable, COMPAS considers compliance checking in three main stages:

- i. Compliance verification of business process models (static verification at design time)
- ii. Compliance monitoring of the running instances (dynamic verification at runtime)
- iii. Offline monitoring of business process executions

These stages are complementary to ensure compliance. In this deliverable we propose a solution for the first stage, which can be seen as “preventive compliance management”. Runtime compliance monitoring and offline monitoring are out of scope of this deliverable.

1.2. Document overview

This deliverable is structured as follows: In Section 1 we describe the purpose and scope of this deliverable and introduce relevant terms and used abbreviations. In Section 2 we position the approach described in this deliverable within the overall COMPAS approach in terms of architecture and the conceptual model. We also discuss how the deliverable is related to other deliverables and work packages. Section 3 forms the main part of this deliverable. This section describes a design time methodology to assure compliance of a business process. The methodology integrates the compliance requirement formalization concepts developed in WP2 with (1) the process structure reuse approach developed in WP4, and (2) with concepts for formal verification developed in WP3. In Section 4 we summarize the deliverable, discuss limitations of the presented approach and characterize future work.

1.3. Definitions and glossary

The most important terminology concerning the COMPAS project is listed on the public COMPAS Web-Site [D7.1] available at <http://www.compas-ict.eu>, section “terminology”. This helps to make the overall COMPAS approach more comprehensive for the general public.

In the following the definitions of terms valid only in the scope of this deliverable (and therefore not listed on the public COMPAS Web-Site) are specified. To offer a self-contained deliverable, general terms of the COMPAS terminology are copied here.

Compliance requirement: A Compliance requirement is a constraint or assertion that results from the interpretation of compliance sources.

Compliance source: A compliance source can be a law, regulation, policy, contract, etc.

Compliance risk: The risk of impairment to the organization’s business model, reputation and financial condition (resulting) from failure to meet compliance requirements.

Control: A statement that describes the restraining or directing influence to check, verify, or enforce rules to satisfy one or more compliance requirement - at the business level.

Compliance rule: A Compliance rule is an operative definition of a compliance requirement which formally describes a control.

Compliance fragment: A Compliance fragment (i.e., a process fragment for compliance) is a connected process structure that can be used as a reusable building block to ensure a consistent specification of compliance regarding a business process. Compliance fragments can be used to implement a compliance rule in terms of activities and control structures.

Compliance target: A Compliance target is a generic specification, such as a business process, or a compliance fragment, which is a target of compliance requirements.

Compliance request: A user (compliance or business expert) can issue a compliance request to check whether a set of compliance targets conforms to a set of applicable compliance requirements. The purpose of a compliance request is to identify if and how a process can or should be changed to make it compliant.

Technical control: A technical control describes the restraining or directing influence to check, verify or enforce rules to satisfy one or more compliance requirements - at the implementation level.

1.4. Abbreviations and acronyms

AD	(UML) Activity Diagram
BP	Business Process
BPEL	Business Process Execution Language
BPM	Business Process Management
BPMN	Business Process Model and Notation
CEP	Complex Event Processing
CRL	Compliance Request Language
CRLT	Compliance Request Language Tools
CRR	Compliance Requirements Repository
CTD	Contrary-To-Duty
CTL	Computational Tree Logic
DSL	Domain-specific Language
EPC	Event-driven Process Chains
LTL	Linear Temporal Logic
SD	(UML) Sequence Diagram
SOX	Sarbanes-Oxley Act
UML	Unified Modelling Language
UUID	Universally Unique Identifier
WP	Work Package
XML	eXtensible Markup Language

2. Integration into the overall COMPAS architecture

In this deliverable we describe a methodology for compliant business process design, which integrates concepts from WP2, WP3 and WP4. Therefore, the components in the overall COMPAS architecture which implement and support these concepts are also relevant for this deliverable. To position this deliverable within the overall COMPAS approach we briefly sketch the solution concept in the following, based on the COMPAS conceptual model. Subsequently, we take a closer look at the relevant components of the overall COMPAS architecture.

The COMPAS consortium has agreed on a conceptual model (listed on the public COMPAS Web-Site [D7.1]) that captures the important concepts of the overall approach. The conceptual model is also shown in Figure 1. This model includes concepts for compliance checking during design time, runtime as well as offline monitoring of processes, services

software framework provides an integration of the fragments with the process model. Additionally, the model repository developed in WP1 provides for access to processes and fragments. The fragments (which can be gathered from the model repository) are synchronized with the process (-fragment) repository developed in WP4. The model repository provides input (i.e., compliance targets) for the Compliance Request Language Tools (CRLT). CRLT select compliance targets and relevant compliance rules and send them to the process verification tools, which are developed in WP3. The components of WP3 check the process against an internalized format of the compliance rules and return the results of the verification step back to CRLT, which present them to the user. Based on these results, further changes on the process might be required. These changes are again performed with WP1 tools (which closes the cycle).

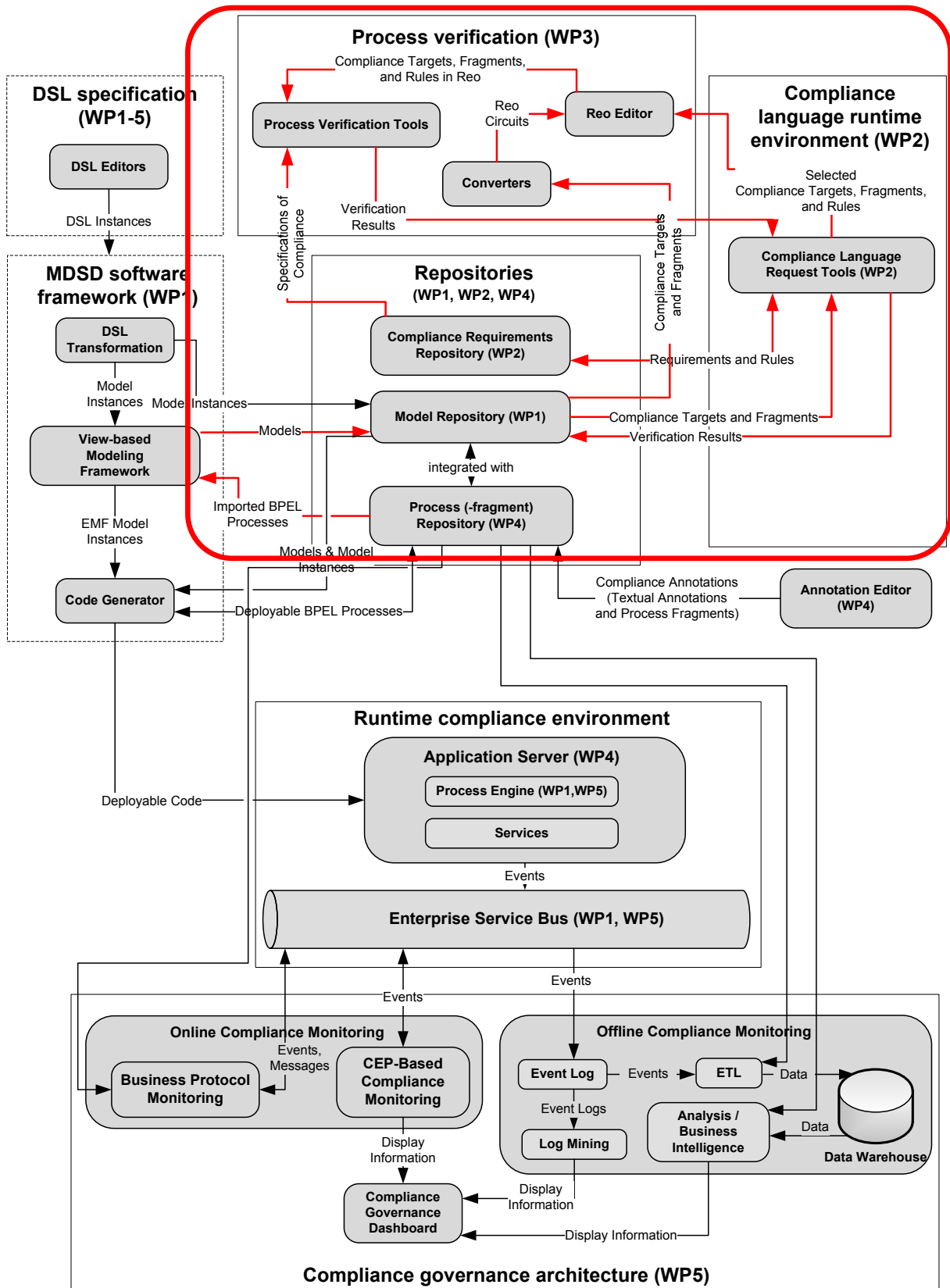


Figure 2 Overall COMPAS architecture

3. Methodology for assurance of compliant business process design

In this section, we elaborate the methodology for assurance of compliant business process design. At first, we discuss the overall proceeding on a high level of abstraction in order to give an overview of the approach, before going into detail afterwards. In particular, we discuss the concepts for definition and formalization of compliance requirements which have been developed in WP2, as well as their relation to the concept of compliance fragments which has been provided by WP4. After a discussion of the creation of compliance fragments and their integration into a process, we show how the concepts and tools for process verification developed in WP3 contribute to our solution for design time compliance management.

3.1. Overall proceeding

The process model shown in Figure 3 illustrates the principle for assurance of compliant process design focusing on a single requirement. We use this process model within this section to show the interrelation of the particular steps that have to be performed in our approach. The initial task in the process is the definition of compliance requirements, which are derived by experts from compliance sources such as SOX or Basel II. Subsequently, these requirements are formalized which allows a verification of the process in the final stage to give a proof of its compliance with the requirements. Particular process structures related to compliance, however, may need to be integrated as some compliance requirements demand for changes of a process. Therefore, we discuss the search and creation of such process structures, and elaborate how they can be integrated into a process.

To obtain a proceeding that covers management of complete compliance sources this proceeding needs to be extended. In addition, several optimizations of the proceeding are conceivable. For instance, one might want to first verify a process before considering an augmentation of the process with compliance fragments.

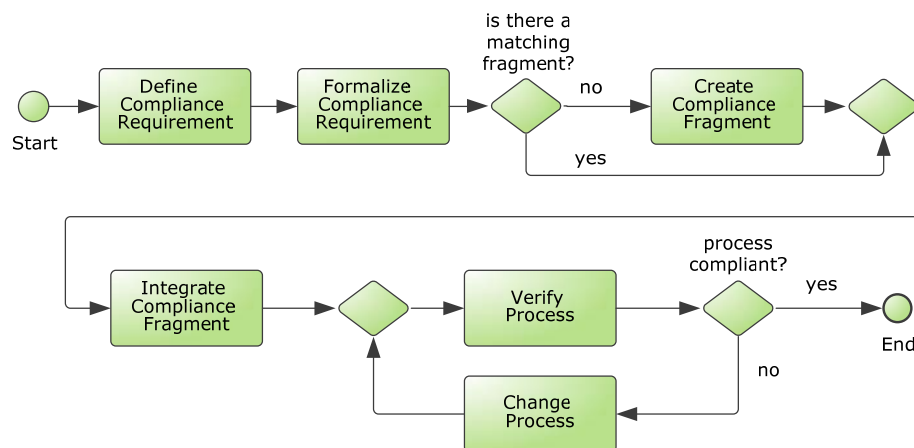


Figure 3 Overall proceeding for assurance of compliant business process design

3.2. Definition of compliance requirements

As an initial step to the overall approach for compliant business process design, the definition of the compliance requirements (Figure 4) also involves the identification and internalization of relevant compliance requirements originating from internal or external sources in various abstraction levels. Based on the conceptual model discussed in Section 2, this task results in a definition of compliance relevant concepts (in particular: compliance requirements, risks, sources and controls) and their internal and external relationships. These concepts are stored and organized in a persistent storage, i.e., the Compliance Requirements Repository (CRR), which also enables their reusability [D2.6]. Besides organizing compliance constraints in a repository, it is important to handle their evolution. These concepts are maintained as compliance requirements and may evolve or change. For instance, the organization may be required to comply with new regulations, standards, etc. Storing compliance requirements in a repository along with necessary metadata helps maintaining their backward and forward traceability to relevant compliance targets.

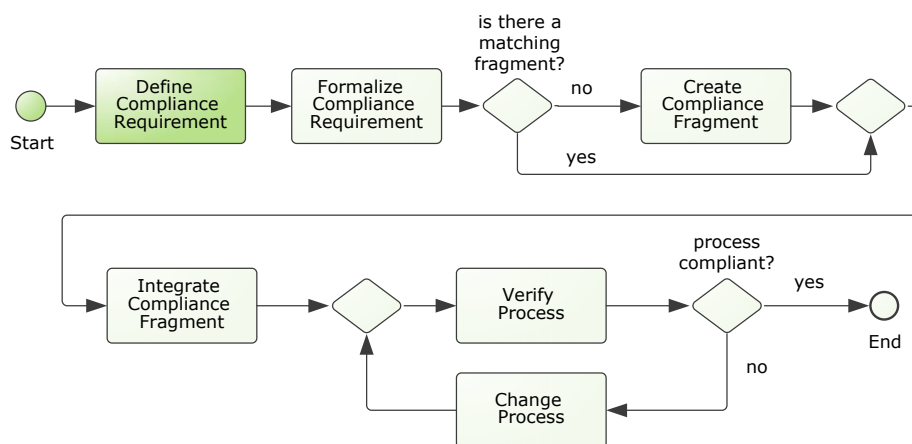


Figure 4 Definition of compliance requirements

The definition and maintenance of compliance data is performed by the *compliance experts* (with close collaboration with the *business experts*). Compliance experts use CRLT to define, store and maintain compliance data and their internal relationships, as well as their external relationships with the compliance targets. Figure 5 shows an interface of the CRLT exemplifying the definitions of compliance requirements, compliance sources and their structural relationships. The details concerning the use of CRLT for the definition of compliance data as well as the other features supported are described in [D2.6].

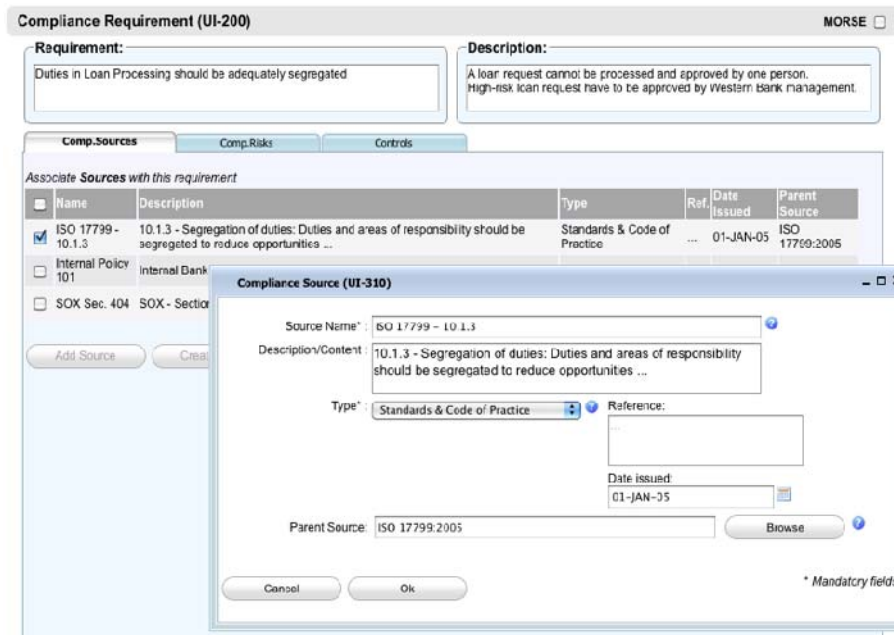


Figure 5 Defining compliance requirements and sources in CRLT

3.3. Formalization of compliance requirements

One of the key aspects in the overall approach presented in this deliverable is grounded on the notion of formal process verification of the business process models (that are enriched by compliance fragments) against formal compliance rules. To enable automated verification, compliance requirements should be formally specified by using a logical language that is based on a formal foundation. Hence, one of the important steps in the overall approach is the transformation of high-level and abstract compliance requirements/controls into formal compliance rules (“Formalize Compliance Requirement” in Figure 6).

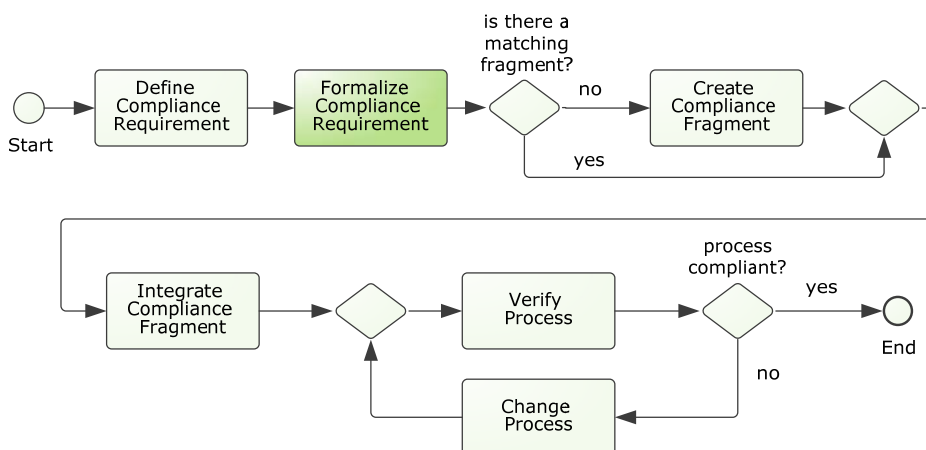


Figure 6 Formalization of compliance requirements

In [D2.2], we have identified the languages that can be used as the basic building blocks for the CRL and performed a comparative analysis between these candidate formalisms (published in [ETH+10a]). We have analyzed a wide range of compliance legislations and frameworks and examined a variety of relevant works on the specification of compliance

requirements. We have identified structural patterns of frequently recurring compliance requirements imposed on the business processes. In particular, we examined Deontic logic (e.g. [SGN07]) and temporal logic (e.g. [LMX07]) families, which have been intensively discussed in the literature as a basis for such a specification language. In our framework, we mainly rely on temporal logic for representing compliance rules. Our choice is justified by the fact that the system property specification using temporal logics is a mature field supported by efficient verification tools tested and applied in practice for over 20 years. Among the formalisms within temporal logic family, we prefer Linear Temporal Logic (LTL) [Pnu77] to Computational Tree Logic (CTL) mainly due to its simplicity, intuitiveness and compositionality of reasoning [Var01].

One of the main problems of the temporal logic family in general is that the logical formulae are difficult to write and understand for users. This issue represents one of the main obstacles for the utilization of advanced verification and analysis tools associated with these languages. To help to manage these difficulties, we adapted property specification patterns as an integral part of CRL [D2.3]. The patterns are based on Dwyer’s property patterns and are high-level abstractions of frequently used logical formulae, which help non-technical users to abstractly represent desired properties and constraints. In addition to the original patterns introduced in [DAC98] we have introduced “compliance patterns” to capture recurring requirements in the compliance context. Figure 7 presents the conceptual model for the pattern expressions used within the CRL.

As shown in Figure 7, an *expression* comprises *property specification patterns* (*patterns* for short), *operands* and *scope*. Expressions can combine multiple (sub) expressions by using Boolean operators. For example, the expression “((P *Precedes* Q) *And* (R *Exists*)) *Globally*” comprises two sub-expressions, where *Precedes* and *Exists* indicate patterns; *P*, *Q* and *R* are operands; *And* represents the conjunction Boolean operator, and *Globally* forms the scope of the expression. *Operands* take the form of business process constructs, such as activities of a process or a compliance fragment, and their attributes. For example, the expression “CreateOrder *LeadsTo* ApproveOrder” has two operands connected via the *LeadsTo* pattern. *Scope* defines a starting and an ending state for an expression. Please refer to [D2.4] for the details regarding the patterns and relevant expression constructs.

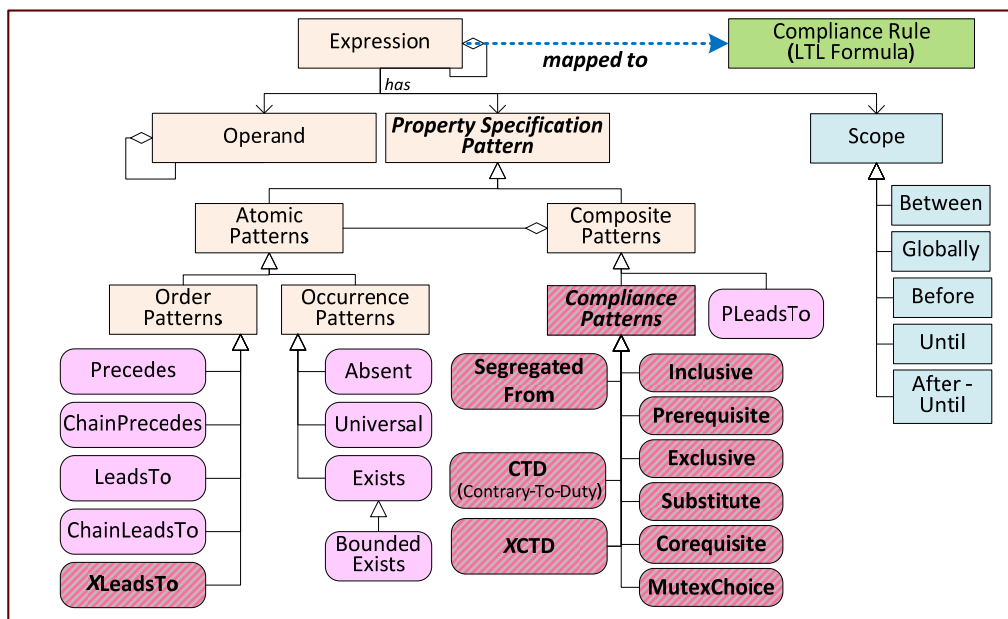


Figure 7 Conceptual model for pattern expressions within CRL

Having internalized the compliance constraints and defined concrete requirements, compliance risks, sources and controls (Section 3.2), a *compliance expert* visually represents the controls as pattern expressions by using the Rule Modeller. Rule Modeller is a standalone component of the CRLT, which allows compliance experts to define pattern-based representations of the controls and automatically generate LTL formulae based on these expressions. [D2.4] presents the details of this component and its relationship with the other components of the COMPAS architecture. The details of the generation rules from pattern expressions to LTL formulae as well as the overall approach for the pattern based representation of compliance requirements are presented in [ETH+10b].

Table 1 presents such patterns applied to the loan origination scenario [D6.1]. The first control is implemented using the newly introduced *SegregatedFrom* pattern that captures the typical compliance requirement, which mandates segregation of duties among different roles and actors. In LTL, *G*, *F*, *U* correspond to the temporal operators ‘always’, ‘eventually’, and ‘until’ respectively. ‘*G*’ denotes that formula *f* must be true in all states of the business process model. ‘*F*’ indicates that formula *f* will be true at a certain state in the future. ‘*U*’ denotes that the first formula is true in all preceding states until the second formula gets true in a certain state. For example, the LTL representation of ‘*P LeadsTo Q*’ is ‘ $G(P \rightarrow F(Q))$ ’, which can be read as: In all states of all runs in the business process, if *P* is true, then *Q* occurs in the future.

Control	Pattern	Compliance Rule in LTL
1 - Customer bank privilege check is segregated from credit worthiness check	CheckCustomerBankPrivilege SegregatedFrom Check Credit Worthiness	$(G(\neg \text{CheckCreditWorthiness } W \text{ CheckCustomerBankPrivilege})) \wedge (G(\text{CheckCustomerBankPrivilege} \rightarrow F(\text{CheckCreditWorthiness}))) \wedge (G((\text{CheckCustomerBankPrivilege.Role}(\text{Role1}) \rightarrow G(\neg(\text{CheckCreditWorthiness.Role}(\text{Role1}))))$
2 - If the loan request exceeds 1M, the Clerk Supervisor checks the credit worthiness of the customer	$((\text{LoanAmount} \geq 1M) \text{ LeadsTo } \text{CheckCreditWorthiness.Role}(\text{"Supervisor"}))$	$G((\text{LoanAmount} \geq 1M) \rightarrow F(\text{CheckCreditWorthiness.Role}(\text{Supervisor})))$
3 - The branch office Manager checks whether risks are acceptable and makes the final approval or rejection of the request.	$((\text{JudgeHighRiskLoan.Role}(\text{'Manager'}) \text{ AND } \text{Approved} = \text{'Yes'}) \text{ Precedes } \text{SignOfficiallyLoanContract.Role}(\text{'Manager'})) \text{ AND } ((\text{JudgeHighRiskLoan.Role}(\text{'Manager'}) \text{ AND } \text{Approved} = \text{'No'}) \text{ Precedes } \text{DeclineDueToHighRisk}(\text{'Manager'}))$	$G((\text{JudgeHighRiskLoan.Role}(\text{'Manager'}) \wedge \text{Approved} = \text{'Yes'}) \vee (\neg \text{SignOfficiallyLoanContract.Role}(\text{'Manager'}) \wedge \text{JudgeHighRiskLoan} \wedge \text{Approved} = \text{'Yes'})) \wedge G((\text{JudgeHighRiskLoan.Role}(\text{'Manager'}) \wedge \text{Approved} = \text{'No'}) \vee (\neg \text{DeclineDueToHighRisk.Role}(\text{'Manager'}) \wedge \text{JudgeHighRiskLoan} \wedge \text{Approved} = \text{'No'}))$

Table 1 Example compliance requirements from the loan origination scenario

Figure 8 shows how the controls and pattern expressions listed in Table 1 can be visually represented with the Rule Modeller.

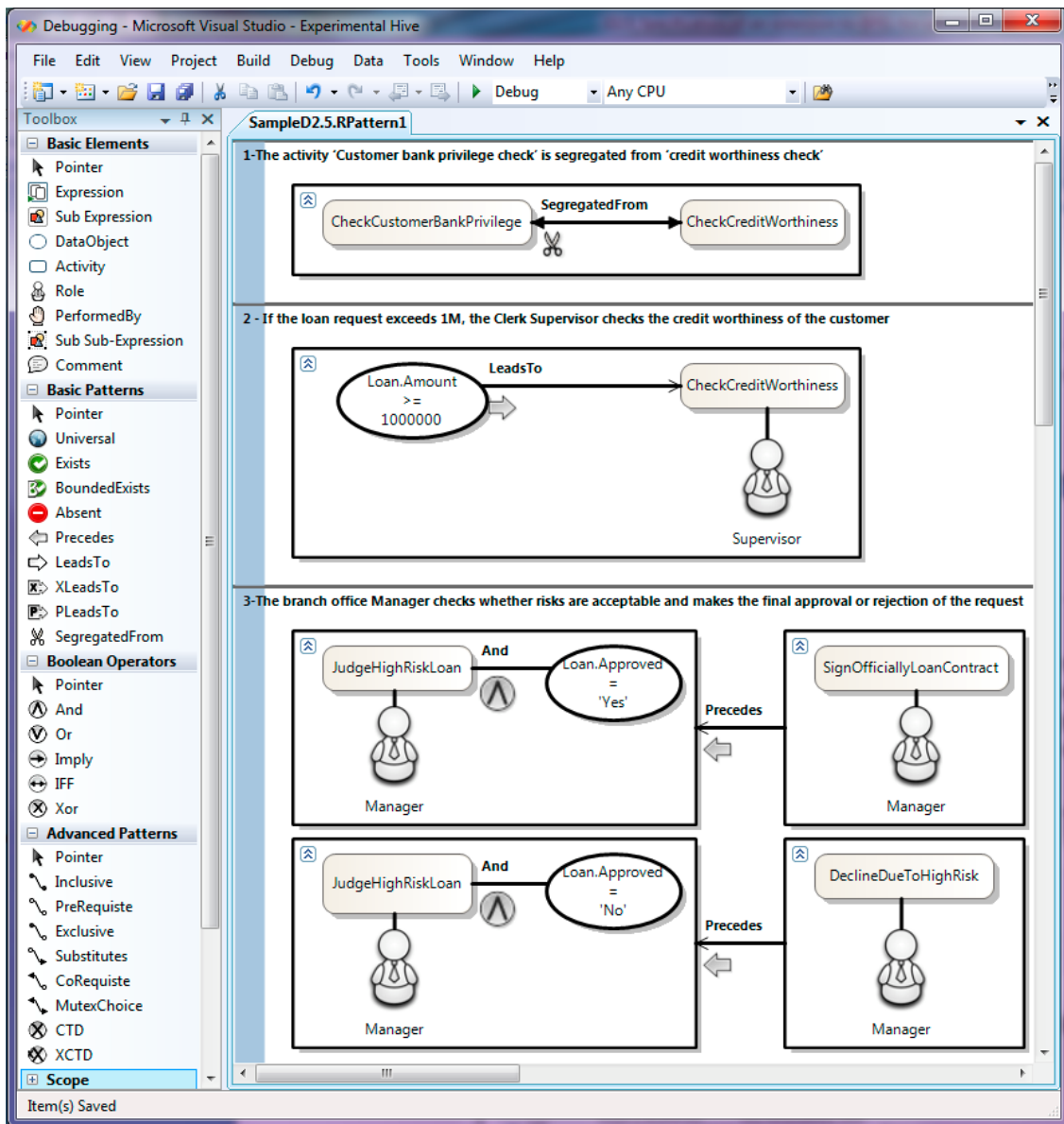


Figure 8 Rule Modeller

CRLT also has a web-based component, which allows end users to formulate compliance requests by selecting compliance targets and relevant compliance requirements for process verification. Figure 9 shows an example where compliance targets in the model repository and associated compliance sources are selected to filter and retrieve compliance data relevant to the selection.

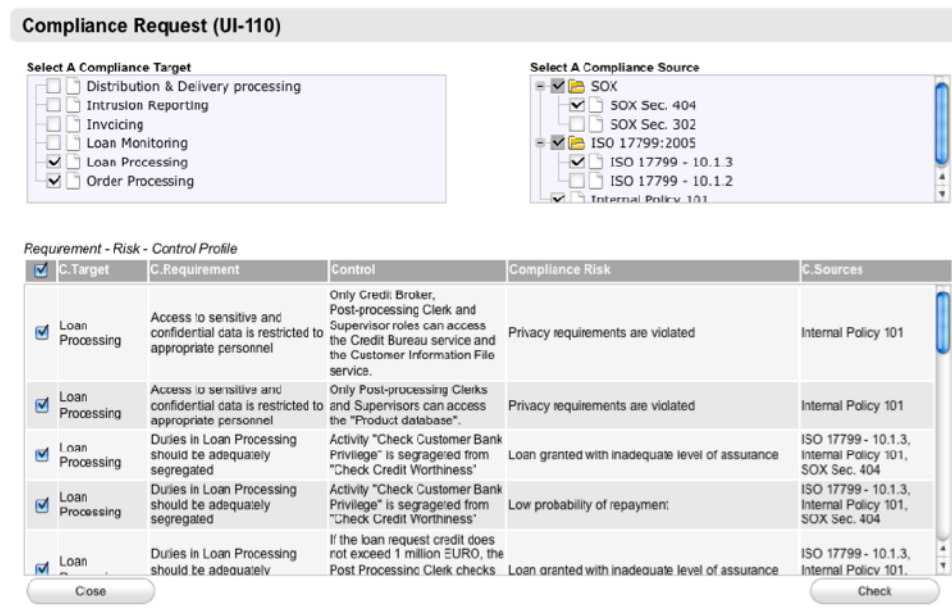


Figure 9 Creating compliance requests

The ongoing integration with the Process Verification toolkit (WP3, see Section 3.6) for process verification is achieved through a group of asynchronous Web service calls. The ‘check’ button (located in the right-bottom corner of Figure 9) packages the compliance request containing the BPEL representations of the compliance targets (i.e. process models) and relevant formal compliance rules specified in LTL, and invokes the Web service to transfer the request to the Process Verification toolkit. After the process verification is performed, the toolkit returns the verification results, listing the rules that have been checked and whether they are satisfied or not. Figure 10 presents one of the user interfaces from the CRLT reflecting how the results of the compliance check are communicated to the business or compliance expert. The user interface exemplifies the case that the first control given in Table 1 is violated.

Compliance Check (UI-120)							
Compliance Target			Check Result		Degree of Compliance		
Loan Processing					84%		
Compliance Profile Details							
Control ID	C.Requirement	CONTROL	C.Risk	C.Source	Weight	Result	
1.1	Duties in Loan Processing are adequately segregated	Activity "Check Customer Bank Privilege" is segregated from "Check Credit Worthiness"	-Loan granted with inadequate level of assurance - Low probability of repayment	ISO 17799 - 10.1.3, Internal Policy 101, SOX	5 - Very High	Activities performed by the same role... Check Details	
ID	Description.	Statement	Type	Design-Time	Run-time	Satisfied?	Result Desc. / Remedy
1	Check if activity exists	F (CheckCustomerBankPrivilege)	LTL	√			
2	Check if activity exists	F (CheckCreditWorthiness)	LTL	√			
3	Check if activities follow each other	G((CheckCustomerBankPrivilege OR (!CheckCreditWorthiness UNTIL CheckCustomerBankPrivilege)))	LTL	√			
4	Check if activities follow each other	G(CheckCustomerBankPrivilege THEN F (CheckCreditWorthiness))	LTL	√			
5	Check if activities are assigned to different roles	G((CheckCustomerBankPrivilege.Role(Role1) THEN G(!CheckCreditWorthiness.Role(Role1)))	LTL	√			Activities performed by the same role
6	Check if activities are assigned to different actors	G((CheckCustomerBankPrivilege.Actor(Actor1) THEN G(!CheckCreditWorthiness.Actor(Actor1)))	LTL		√		This rule can be checked when Runtime information is available
1.2	Duties in Loan Processing are adequately segregated	If the loan request credit does not exceed 1 million EURO, the Post Processing Clerk checks the credit worthiness of the customer.	Customer initial credit worthiness check is segregated from post check, which is performed by supervisor role	-Loan granted with inadequate level of assurance - Low probability of repayment	3 - Moderate		

Figure 10 A user interface with compliance check results displayed to the user

3.4. Search and creation of a suitable compliance fragment

In the previous step of our methodology we obtained a set of formalized compliance requirements (compliance rules). Possibly, these rules demand for particular tasks that have to be performed in a process, for instance a manager has to make an additional approval. As some requirements occur more frequently, a suitable solution might already exist. Therefore, a search for corresponding compliance fragments needs to be made. If this search is not successful, a new compliance fragment needs to be created as depicted in Figure 11. In the following, we discuss the search for compliance fragments as well as their creation in more detail.

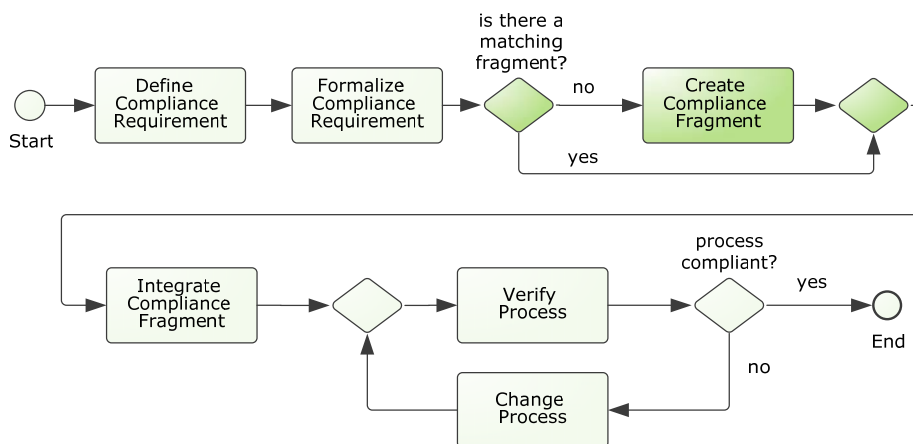


Figure 11 Search and creation of a suitable compliance fragment

In WP4 we developed a repository for the management of compliance fragments [D4.4]. The search functionality for compliance fragments is accessible via Web service interfaces as well as via a Web based user interface, as presented in Figure 12.

The screenshot displays the 'Fragmento for COMPAS' web interface. At the top, there is a browser tab and the application logo. Below the logo is a navigation bar with buttons for 'Start', 'Artefacts', 'Relations', and 'Locks'. A secondary navigation bar contains buttons for 'Manage', 'Search', 'Create', 'Check Out', 'Check In', and 'Transform'. The main content area features five search panels, each enclosed in a dashed border:

- Search in the Description:** A text input field labeled 'Search for contained String:' followed by a 'Search' button.
- Search in the Content:** A text input field labeled 'Search for contained String:' followed by a 'Search' button.
- Search by Type:** A dropdown menu labeled 'Search for Type:' with 'Annotation' selected, followed by a 'Search' button.
- Search by Date:** Two text input fields labeled 'Start Date of Creation:' (with '2008-11-25') and 'End Date of Creation:' (with '2010-11-25'), followed by a 'Search' button.
- Search by Date and Type:** A dropdown menu labeled 'Search for Type:' with 'Container' selected, and two text input fields for 'Start Date of Creation:' (with '2008-11-25') and 'End Date of Creation:' (with '2010-11-25'), followed by a 'Search' button.

Figure 12 Search in Fragmento

The process fragment repository provides certain options to search for a corresponding fragment:

- Search within the fragment description: Our fragment repository allows storing additional metadata for the compliance fragments. The description can either be regular text (containing the fragment's name, description, keywords, usage domain, number of fragment entries and exits etc.). In more advanced scenarios the usage of semantic information could also be considered, e.g. using ontologies. We enable this search option by a full-text search within the description that is stored together with the fragments.

- Search within the fragment content: For instance, such a search can be used to find compliance fragments which use particular services. We enable this search option by a full-text search within the XML code of the fragments.
- Search by date of creation or change: We use this search during the synchronization of the different repositories in order to synchronize changes which have been made in a particular timeframe. Regarding the search for a fragment that corresponds to a particular compliance requirement this option is of minor importance.

In [D4.4] we have described that the process fragment repository can be extended with custom query functions which go beyond the basic search options discussed above. For instance, metadata that is annotated to a fragment (e.g. security annotations in WS-Policy) can be used in a query. Another example: the structure of a compliance fragment could be used as a search criterion. A fragment skeleton could either be sketched by a user, or possibly even be generated out of a given formal compliance rule. A graph matching algorithm could then find the fragment candidates that match this criterion. The development of such search functionality is, however, outside of the scope of the project. In order to enable interested researchers to elaborate on this concept the process fragment repository will be published as open source under Apache licence [Fra10]. In addition, we encourage students to do their Master thesis on this research topic [Pos10].

When the search finally results in a compliance fragment that can be used to realize a compliance requirement, then it is beneficial to establish a link between the compliance fragment and the corresponding compliance requirement. In case, the requirement has to be realized in another process later, the corresponding fragments are already known (i.e., linked) and a time-consuming search is not required anymore. The repositories developed in COMPAS already provide the functionality to establish such a link.

In case there is no matching compliance fragment found, then it has to be created. In [SLM+10] we proposed two different approaches: the bottom-up approach and the top-down approach.

In the bottom-up approach a fragment is created from scratch. This approach can be understood as cooperative task: a compliance expert specifies the requirements the fragment has to meet, and a process designer implements a process structure that realizes the requirements. The tools for process verification developed in WP3 (discussed in Section 3.6) can also be used to verify a single fragment against formal rules, which is quite useful to check if the design of the fragment complies with the rules. However, checking a fragment against the rules is not sufficient to proof compliance of the augmented process, as mistakes which might be made during integration might violate the original design intention. This case and a solution are discussed in Section 3.5.

In the top-down approach, the process structures related to particular requirements are already contained in a process. For instance, this is the case in a process which is already proven to comply with the given compliance rules. In this scenario, the first step is to identify which parts of the process belong to the compliance fragment which is contained in the process. The next step is to extract these parts. In [SLS10] we propose a view-based technique to extract a compliance fragment from a given process. This technique is also elaborated in more detail in the WP4 deliverable on the classification and specification of reusable process artefact [D4.3]. After extraction from a process, the fragment needs to be generalized to ease later reuse, i.e. particular attributes are parameterized and process-specific data is removed.

All in all, both approaches result in a compliance fragment that can be used to augment a process with compliance. Figure 13 shows a compliance fragment for an approval as an

example. This fragment has one entry, which starts the approval. The fragment has two exits with different semantics. One exit represents the exit in case of acceptance and the other exit represents the rejection case.

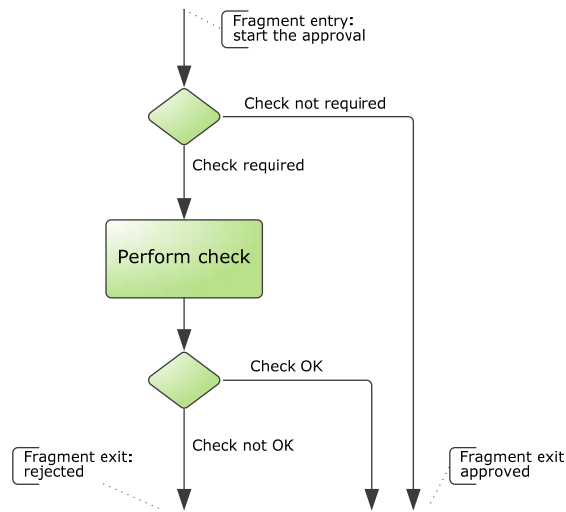


Figure 13 Abstract compliance fragment for an approval [STK+10]

After a compliance fragment has been created, it can be stored in the process fragment repository, so that it is available when other processes also need to comply with the requirement which the compliance fragment realizes.

3.5. Integration of a compliance fragment into a process

The next step in the methodology for assurance of compliance is the augmentation of a process with a compliance fragment (depicted in Figure 14).

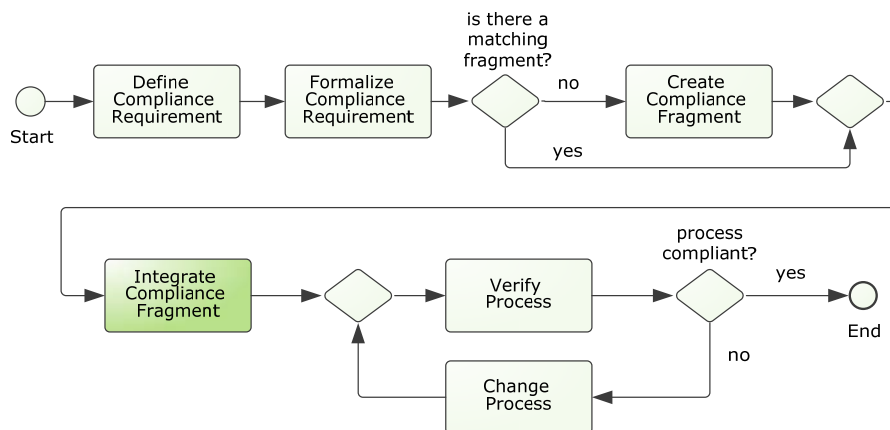


Figure 14 Integration of a compliance fragment into the process

As we have pointed out in [STK+10], the compliance fragment integration step is the major weakness of the compliance fragment approach. This is due to the possibility, that after integration the process design might still violate the compliance requirements. We have identified three major reasons for that:

- i. Wrong positioning: A compliance fragment has to be integrated at a particular place in the process. Let us assume, a requirement that demands for a *final* approval or

rejection of the loan request by the branch office manager (Control 3 in Table 1). The corresponding compliance fragment for this approval might be misplaced in a path that is not always executed or there might be another approval in a later stage in the process.

- ii. Wrong concretization: During integration the parameters of a compliance fragment have to be adjusted, according to its usage context. For instance, the compliance fragment used for the final approval might be concretized using a wrong role, e.g. using clerk instead of branch office manager.
- iii. Change of the original fragment design: The compliance fragment is designed to express a particular behaviour. In case the modelling environment, however, does not enforce it, the original design of the fragment might be broken. For instance, the ordering of the activities of the fragment might be changed, activities added or removed, or control links be changed.

In other words, by integrating a compliance fragment into a process, there is no evidence or proof yet, that the resulting process design is actually compliant. Therefore, the process verification step which follows the integration is crucial for an overall solution.

In Section 3.4 we have described that a compliance fragment might be rendered somewhat abstract in order to ease its later reuse. This abstraction from details necessitates a concretization in order to fit the fragment into the context in which it is actually applied. The concretization steps are performed during the integration of a compliance fragment into a process. In Figure 15 such a concretization is exemplified, based on the compliance fragment for approval. In this example the concretization includes the following changes:

1. Activation condition: In its abstract form the condition states whether a check is required or not. For usage of this fragment in the loan originating scenario this condition is concretized to a formula that indicates whether a risk is low or not (more precise: if loan request is below 1 M Euros).
2. Activity for checking: In its abstract form, the activity states that a check is performed. Regarding the loan originating scenario, this activity is concretized to judge a loan request, based on particular input data available in the process context. Furthermore, the role of the human who has to perform the task is specified (Manager).
3. Decision condition: Based on the output of the approval task a condition needs be specified that defines whether the loan is approved or rejected. This might also include the usage of business rules in particular cases.

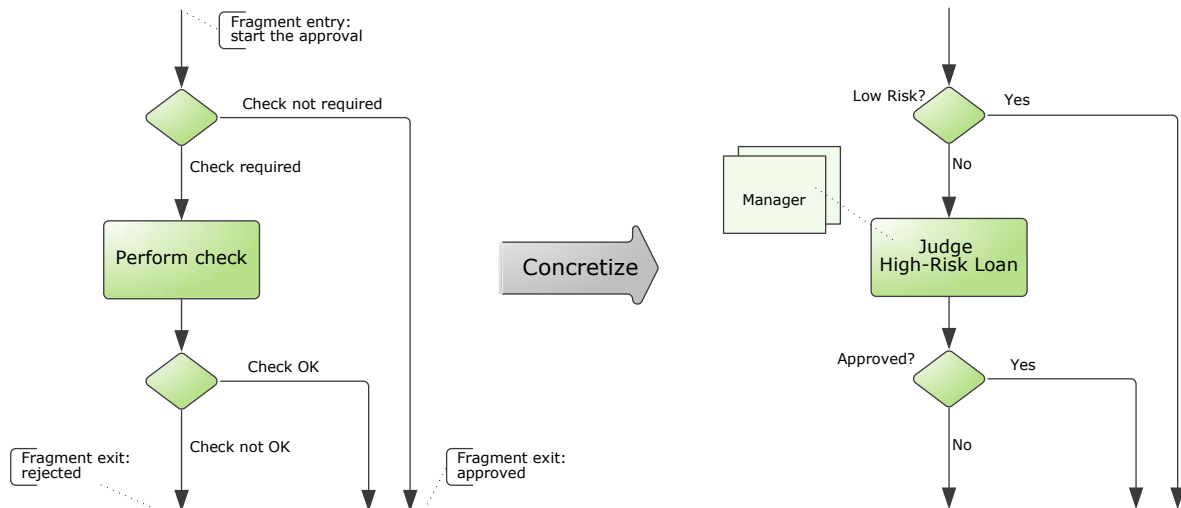


Figure 15 Abstract compliance fragment for approval and its concretization for a concrete usage scenario [STK+10]

Listing 1 shows the conceptual algorithm that contains the different technical operations that have to be performed for integration of a compliance fragment into a BPEL process.

```

00 //Integration of a compliance fragment into a process
01 Function integrate (fragment F, process P)
02     wireEntries (F,P)
03     wireExits (F,P)
04     concretizePlaceholders (F,P)
05     mergeContext (F,P)
06     mergeHandlers (F,P)
07     injectProcessStructures (F,P)
08     return P
09 end

```

Listing 1 Conceptual algorithm for compliance fragment integration

During integration the fragment has to be “wired” with the process. First, the entries of a fragment are wired with the process (`wireEntries(F,P)`, line 02). In other words, control links between activities in the process and in the fragment are being established. Analogously, the exits of a fragment have to be wired with the process as well (`wireExits(F,P)`, line 03). Then, the abstract placeholders in a fragment have to be concretized to fit the fragment into the process (`concretizePlaceholders(F,P)`, line 04). As the fragment might have its own context information (data types, correlation sets etc.) this information needs to be merged with the context of the process. As particular handlers might be defined for the fragment (such as fault or compensation handlers), these structures also need to be merged with the process context (i.e. with the parent `<scope>` in terms of BPEL; `mergeContext(F,P)` and `mergeHandlers(F,P)`, line 05 and 06). Furthermore, the activities and related control structures of the compliance fragment need to be injected into the process document, i.e. copied (`injectProcessStructures(F,P)`, line 07).

The result of the compliance fragment integration is a process model that does not contain any fragment-related extensions such as entries, exits or regions anymore (`return P`, line 08). These constructs are removed during integration. There is one exception: For traceability reasons, the extension for unique identifiers on the constructs remains. This extension allows tracing which parts of the process originally belonged to which compliance fragment and its particular version. The usage of unique identifiers on constructs does not change the semantics of the process though. For more details on the mentioned BPEL extension please see deliverable D4.2.

The implementation of the conceptual algorithm shown in Listing 1 is ongoing integration work between WP1 and WP4. In particular, we are currently extending the View-based Modeling Framework (VbMF) developed in WP1 [D1.3] with functionality to access and use artefacts stored in the model repository and the process fragment repository (developed in WP4, see [D4.4]). We envision a wizard application that assists and guides the process designer during the augmentation of a process with compliance fragments. Furthermore, we also investigate merging the WSDL interface descriptions and policies which might be attached to a compliance fragment.

3.6. Verification of the process

To detect the violation of compliance rules in business process models, we use Process Verification Tools developed in WP3 [D3.2, D3.3]. In this section, we describe how these tools are integrated with CRLT and how they are used to achieve process compliance by design.

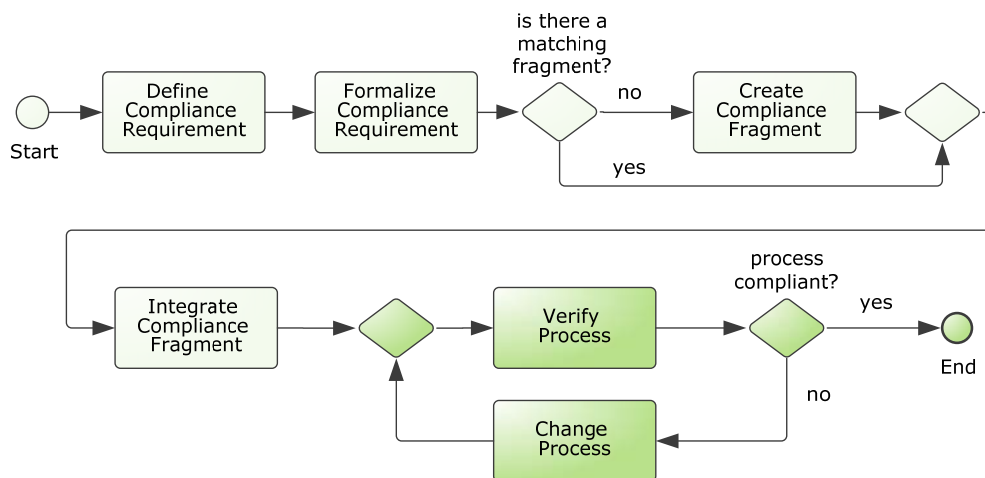


Figure 16 Verification of the process

As presented in Figure 16, after the integration of a developed compliance fragment into a process model, the latter is verified against formalized compliance requirements specified in CRL to ensure that the integration was carried out correctly. The verification process is usually iterative: in case a property is violated, we expect the designer to refine the process and verify it again. The Process Verification Tools help the process designer to locate an error by providing execution traces that lead to the violation of the requirement.

The technical integration of the CLRT and a model repository with the verification tools is accomplished through an asynchronous Web service call. We have implemented a Web service that provides two basic operations, namely, `int AcceptReq(IncomingReq acceptReq)` and `VerificationRes GetVerificationRes(int id)`. The first operation

accepts a request for the process verification and assigns a unique identifier to this request. The second operation is used to retrieve the results of the verification given an identifier. The requests are submitted by the process designer that manages compliance requirements with CRLT. The Web service expects as input a data structure with the elements shown in Table 2.

Element	Type	Description
modelID	int	An identifier of a process specification in the model repository.
modelType	string	The format of the process specification, BPEL, BPMN, UML AD, UML SD.
requestDateTime	dateTime	Time and data stamps of the request submission.
requestID	int	An identifier of the request.
ruleID	int	An identifier of the rule in the compliance rule repository.
ruleStatement	string	A CRL compliance rule to be verified.
ruleType	int	The type of the formal logic used to specify the property with the following values: 1 for LTL, 2 for CTL, 3 for μ -calculus and 0 for other formats.

Table 2 Structure of the verification request

The submitted request is stored in a request repository which can be accessed by a technical expert. The overall process is shown in Figure 17. The technical expert can manage pending requests with the help of the verification request management tools shown in Figure 18 (general view) and Figure 19 (details of a specific request).

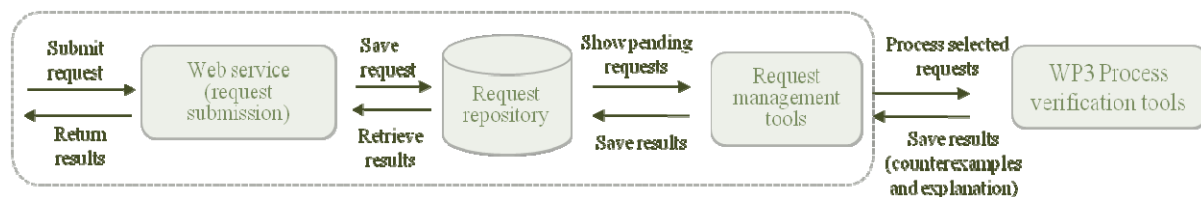


Figure 17 Integration with the Process Verification Tools

No	Request Id	Request Title	Request Date	Deadline	Status	Contact Email	View
1	1		Unknown	Unknown	Not Started		
2	2		Unknown	Unknown	Not Started		
3	653		1970-01-01 00:00:00.0	2004-02-10	Not Started		
4	653		1970-01-01 00:00:00.0	2004-02-10	Not Started		
5	653		1970-01-01 00:00:00.0	2004-02-10	Not Started		
6	653		1970-01-01 00:00:00.0	2004-02-10	Not Started		
7	653		1970-01-01 00:00:00.0	2004-02-10	Not Started		
8	55		2012-05-27 00:00:00.0	2010-05-27	Not Started		
9	55		2012-05-27 00:00:00.0	2010-05-27	Not Started		
10	55		2012-05-27 00:00:00.0	2010-05-27	Not Started		
11	55		2012-05-27 00:00:00.0	2010-05-27	Not Started		

Figure 18 Request management tools: repository

Figure 19 Request management tools: verification request details

The technical expert chooses a request from the repository and imports the process model to the Process Verification Tools. These tools essentially consist of a set of integrated plug-ins that provide the functionality for converting, editing, animating, annotating, simulating and model checking process models. As high-level models provided by a process designer often do not contain all the information necessary for the automated process verification, we assume that this environment is used by the technical specialist to refine these models. The imported process models need to be refined and the compliance rules have to be transformed to a format acceptable by a specific model checking tool chosen to verify a given property.

Figure 20 shows a Reo counterpart for the approval fragment considered in Section 3.4 and 3.5. In this model, an abstract activity *Perform check* is represented as a buffer while conditional gateways correspond to nodes with outgoing filter channels. More details on process formalization can be found in [D3.1] and [D3.2].

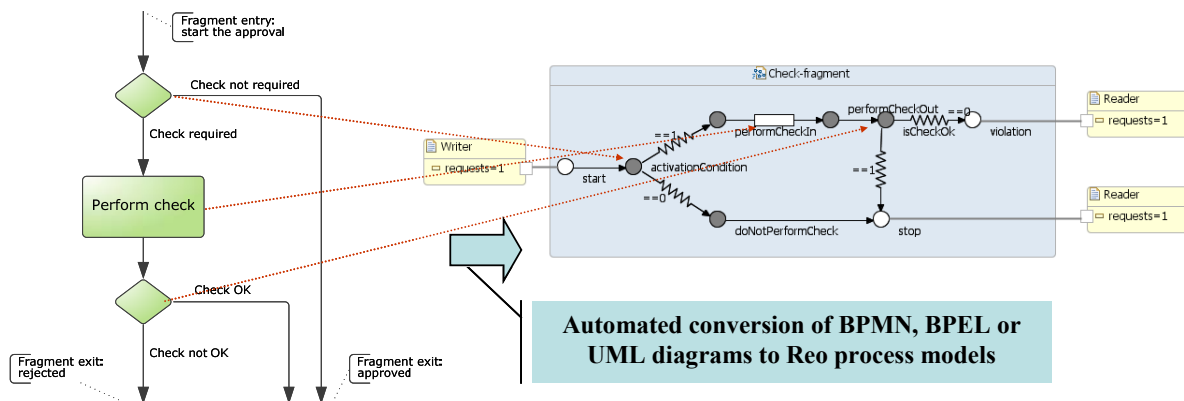


Figure 20 Process formalization: compliance fragment is converted to Reo [STK+10]

Currently, three model checking tools are supported for Reo: Vereify <http://www.verifyfy.de/>, mCRL2 <http://www.mcrl2.org> and PRISM <http://www.prismmodelchecker.org/>. Vereify is a tool that can check properties specified in LTL and CTL-like logics and can be used for control flow analysis. Among its advantages are its compatibility with CRL (which currently deals mostly with LTL-like formula) and the ability to visualize counterexamples in a user-friendly manner by showing them on Reo models using flash animations. However, data specification supported by Vereify is not elaborate enough to enable the verification of data-dependent compliance rules. Such rules can be analyzed with the help of the mCRL2 toolset.

mCRL2 specification language and the corresponding toolset were developed by the University of Eindhoven and represent a powerful means for large-scale system verification. We developed a plug-in for automatic generation of mCRL2 specifications from Reo process models [KKV10], annotated, if necessary, with data and time constraints. For example, Figure 21 shows the model of the dataflow in the compliance fragment for approval, where the input data domain is described by a sort $e1(activated: Bool, amount: Nat)$ which indicates whether the approval is activated and provides the requested loan amount. Data constraints in a format understandable by mCRL2 (e.g., $amount(e1(d)) > 1000000$) are used as annotations to graphical Reo models and specify process dataflow branching conditions.

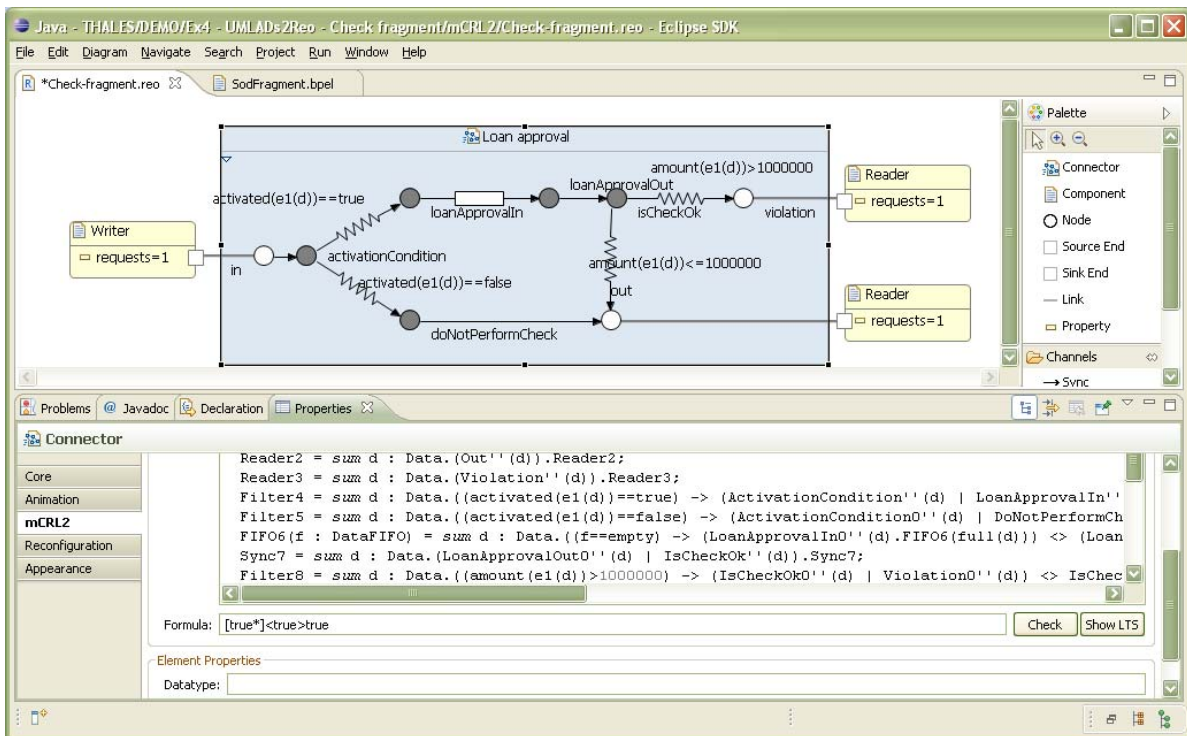


Figure 21 Formal process model refinement: data constraints

Such a model can be used for explicit state space generation or model checking against properties specified in a variant¹ of μ -calculus. This format subsumes temporal logics LTL and CTL and allows us to formally express compliance rules with time and data-aware conditions. For example, a compliance rule “*if a requested loan amount is higher than 1M, a supervisor authorization must be obtained*” (cf. Control 2 in Table 1) corresponds to the following formula:

$$[true^*.\exists amount : N.loanRequest(amount) \wedge (amount > 1000000)]\mu X[\overline{authorization}]X$$

This formula literarily states that for a loan request with the amount exceeding 1M Euro the authorization activity is unavoidable.

Finally, the PRISM model checker can be used for the verification of probabilistic and quantitative properties of a Reo process model. More detailed study of the application of this tool to compliance analysis constitutes our future work.

After processing a verification request, the technical expert saves the results and the intermediate models in the request repository. He can annotate the results with additional

¹ http://www.mcr12.org/mcr12/wiki/index.php/Language_reference/Modal_formulas

information, e.g., explaining why the property does not hold or why the verification tools were not able to analyze the model. Counterexamples found by the model checking tools can help the designer to understand why the property violation occurs in the composed process (e.g., detect fragments that are implemented in a wrong way, point out where the wrong integration points or incorrect placements of fragments are). Such kind of extra information can be introduced to the repository using the Web interface shown in Figure 22. Using this interface, the technical expert can also change the status of the request. Initially, the status is set to 0 (the request has not been processed), then it can be set to 1 (verification in progress), and, finally to 2 (verification has been completed).

Verification Result

Rule Statement:
Customer bank privilege check is segregated from credit worthiness check

Model: [model](#)

Rule Type	<input checked="" type="radio"/> LTL <input type="radio"/> CTL <input type="radio"/> muCalculus <input type="radio"/> None	Model Checker Tool	<input type="radio"/> mCRL2 <input type="radio"/> Vereofy <input checked="" type="radio"/> None
Modified Model	<input type="text"/> <input type="button" value="Browse..."/>	Reo file	/ufs/behnaz/runtime-New_co <input type="button" value="Browse..."/>
Counter Example	<input type="text"/> <input type="button" value="Browse..."/>	Formalized Rule	G((CheckCustomerBankPrivilege.Role(Role1) -> G(!(CheckCreditWorthiness.Role(Role1))))
Result	Activities are performed by the same user.		
Description	See the counter example.		
Counter Example Flash File	<input type="text"/> <input type="button" value="Browse..."/>	Status:	<input type="radio"/> In Progress <input checked="" type="radio"/> Done

[Back](#)

Figure 22 Request management tools: verification results

The process designer can check the status of a certain request and, once the request has been processed, obtain the results. The verification results consist of a data structure with the elements shown in Table 3.

Element	Type	Description
annotation	string	An optional field, which can be used for providing any extra information about the verification process, i.e., specify that the analysis failed because of the lack of necessary details in the model, incorrect formula, etc.
counterExFlashFile	string	A flash animation file that shows a counterexample found by a model checking tool, i.e., a process execution trace that violates a given requirement.

Element	Type	Description
counterexampleFile	string	A file with a counterexample.
explanation	string	A textual explanation of the verification results by a technical expert.
formalizedRule	string	A logic formula for the CRL compliance rule after rewriting by a technical expert. Such rewriting may be necessary to adopt the formula to the terminology in a formalized process model or make it compatible with the format acceptable by a chosen model checking tool.
modelCheckerTool	int	A model checking tool used to verify the process: 1 for mCRL2, 2 for Vereofy, 3 for none.
ruleType	int	The type of the formal logic used to specify the property with the following values: 1 for LTL, 2 for CTL, 3 for mu-calculus and 0 for other formats.
modifiedModel	string	A modified process model used for verification. Sometimes it may be necessary to refine the submitted model to be able to analyze it automatically.
reoFile	string	A formalized process model in Reo.
requestID	int	An identifier of the request.
result	int	The result of the property verification which specifies whether the property holds or not.
revisionDate	dateTime	The date of the last modification.
ruleAnnotation	string	A comment or explanation of the formalized compliance rule.

Table 3 Structure of the verification results

Apart from process model checking, formalized Reo process models can be used for model-based test generation [Tre08]. In this case, generated tests may assure the compliance of an actual system implementation rather than just the designed model. For example, in the aforementioned scenario, at least four test instances should be generated, with and without activated check conditions and loan requests with two amounts: one exceeding 1M, and one not exceeding 1M. Model-based test generation tools such as JTorX <http://fmt.cs.utwente.nl/redmine/wiki/jtorx/> are compatible with the generated mCRL2 specifications and can be easily employed in our framework. Currently we are investigating this research direction.

4. Conclusion

4.1. Summary

In this deliverable, we have integrated the different concepts developed in the WP2, WP3 and WP4 and thereby created a comprehensive and consistent approach for compliance management during design time. We have shown how the extensions to BPEL specified in WP4 [D4.2] can be used to adapt a process to comply with particular requirements, and we have discussed how a proof can be made about whether the adaptation has been made in a compliant manner or not, based on the compliance language developed in WP2 and the verification concepts developed in WP3. We have learned that process adaptation for compliance requires on the one hand a fine-granular concept for process structure reuse, but on the other hand it also necessitates formal methods to give a proof of a compliant design.

4.2. Limitations of the approach

It has to be said that the approach described in this deliverable is limited, when positioning it within compliance management in more general. Our work with the usage scenarios defined by our industry partners (cf. [D6.1]) has revealed that compliance affects almost any part of the business, which includes the various different IT systems, business processes (automated and non-automated), and the people who are involved in them. The scope of the presented approach is focused on compliance requirements regarding automated business processes. In addition, the presented approach discusses design time aspects; elaboration of runtime aspects is ongoing work in WP5 though. Furthermore, not all compliance requirements regarding a business process can be implemented using process fragments. For instance, non-functional requirements related to security need to be specified using the Security DSL [D5.4] to instrument the security components.

Nevertheless, the proposed methodology represents an important part in an overall solution to manage compliance, as it provides a way to check a process for compliance, and to augment it with compliance if required.

4.3. Outlook

The prototypes and concepts which we develop in COMPAS mainly focus on service orchestration using BPEL [OASIS07]. The methodology we have discussed in many of our deliverables (including this one), however, can also be applied to other graph-based process languages such as the BPMN [OMG10], UML Activity Diagrams, or Event-driven Process Chains (EPC) [Kin04]. In parts, the prototypes we develop already consider multiple process languages, e.g. [D3.2] to name an example which also supports the import and verification of process models specified in BPMN and UML Sequence Diagrams.

We are currently working on further integration of the prototypes of the COMPAS architecture. Regarding WP2, the focus is on integration of the View-based Modeling Framework (VbMF) developed in WP1 with CRLT on the one hand, and with the process fragment repository on the other hand, in order to provide an end-to-end solution for handling user-specified compliance requests. The results of this integration work will be reflected in a WP2 prototype delivered in M35 [D2.6], “Implementation of an integrated prototype handling interactive user specified compliance requests in a compliance language”.

Reference documents

4.4. Internal documents

- [DoW] “Description of Work for COMPAS”, 2008-02-01.
- [DA.1] “COMPAS Architectural Walkthroughs and Evaluation Metrics”, 2009-06-08.
- [D1.3] “MDSO software framework for business compliance”, 2009-12-22.
- [D2.1] “State-of-the-art in the Field of Compliance Languages”, 2008-07-31.
- [D2.2] “Initial Specification of Compliance Language Constructs and Operators”, 2009-06-08
- [D2.3] “Design of Compliance Language Run-time Environment and Architecture”, 2009-07-31.

- [D2.4] “Initial implementation of a compliance language”, M30.
- [D2.6] “Implementation of an integrated prototype handling interactive user specified compliance requests in a compliance language”, 2009-12-17, M35.
- [D3.1] “Specification of a behavioural model for services”, 2008-07-31
- [D3.2] “Visual Environment for Service Description”, 2009-07-31.
- [D4.1] “State-of-the-art report on the existing approaches to improving reusability of processes and service compositions”, 2008-12-31.
- [D4.2] “BPEL extensions for compliant services”, 2009-12-31.
- [D4.3] “Classification and specification of reusable process artefacts”, M30
- [D4.4] “Supporting infrastructure – process engine, process artefact repository, process generation tool”, 2009-12-31, M35.
- [D5.4] “Reasoning mechanisms to support the identification and the analysis of problems associated with user requests”, 2009-12-22.
- [D6.1] “Use Case, Metrics and Case Study”, 2008-07-31.
- [D7.1] “Public Web-Site”, <http://www.compas-ict.eu>

4.5. External documents

- [DAC98] M. Dwyer, G. Avrunin, J. Corbett: Property Specification Patterns for Finite-State Verification. Proceedings of the Int. Workshop on Formal Methods on Software Practice, pp. 7-15, 1998.
- [ETH+10a] A. Elgammal, O. Turetken, W-J. van den Heuvel, M. Papazoglou: On the Formal Specification of Business Contracts and Regulatory Compliance. Proceedings of the 4th Workshop on FLACOS. EPTCS, Pisa, Italy, 2010 (to appear).
- [ETH+10b] A. Elgammal, O. Turetken, W-J. van den Heuvel, M. Papazoglou: Root-Cause Analysis of Design-time Compliance Violations on the basis of Property Patterns. Proceedings of the 8th International Conference on Service-Oriented Computing (ICSOC10), USA, 2010 (to appear).
- [FCD+09] F. Daniel, F. Casati, V. D'Andrea, S. Strauch, D. Schumm, F. Leymann, E. Mulo, U. Zdun, S. Dustdar, S. Sebahi, F. de Marchi, M. Hacid: Business Compliance Governance in Service-Oriented Architectures. Proceedings of the IEEE Twenty-Third Int. Conference on Advanced Information Networking and Applications (AINA'09), 2009.
- [Fra10] Fragmento - Fragment-oriented Repository. Online Documentation, 2010. <http://www.iaas.uni-stuttgart.de/forschung/projects/fragmento/start.htm>
- [Kin04] E. Kindler: On the Semantics of EPCs: A Framework for Resolving the Vicious Circle. Proceedings of the 2nd International Conference on Business Process, pp. 82–97, 2004.
- [KKV10] N. Kokash, C. Krause, E. de Vink: Data-aware Design and Verification of Service Composition with Reo and mCRL2. Proceedings of the SAC'10, ACM Press, 2010.

- [LMS05] P. Leach, M. Mealling, R. Salz: A Universally Unique Identifier (UUID) urn Namespace, RFC4122, 2005.
- [LMX07] Y. Liu, S. Muller, K. Xu: A Static Compliance-Checking Framework for Business Process Models. IBM Systems Journal, vol. 46, 2007.
- [OASIS07] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guízar, N. Kartha, C.K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu: Web Services Business Process Execution Language Version 2.0. OASIS Comitee Specification, April 2007, <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>
- [OMG10] Object Management Group: Business Process Model and Notation, Version 2.0, June 2010, http://www.omgwiki.org/bpmn2.0-ftf/lib/exe/fetch.php?id=public%3Areport&cache=cache&media=public:bpmn2.0_dtc-2010-06-04_-_no_markup.pdf
- [Pnu77] A. Pnueli: The Temporal Logic of Programs, Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, Providence, pp. 46–57, 1977.
- [Pos10] A. Poszlovszki: Process Fragment Recognition and Emphasis. Diploma thesis no. 3001, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2010 (to appear).
- [SGN07] S. Sadiq, G. Governatori, K. Naimiri: Modeling Control Objectives for Business Process Compliance. Proceedings of the 5th Int. Conference on Business Process Management (BPM'07), pp. 149-164, 2007.
- [SKL+10] D. Schumm, D. Karastoyanova, F. Leymann, S. Strauch: Fragmento: Advanced Process Fragment Library. Proceedings of the 19th Int. Conference on Information Systems Development (ISD 2010), Springer, 2010 (to appear).
- [SLM+10] D. Schumm, F. Leymann, Z. Ma, T. Scheibler, S. Strauch: Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. Proceedings of the Multikonferenz Wirtschaftsinformatik (MKWI'10), 2010.
- [SLS10] D. Schumm, F. Leymann, A. Streule: Process Views to Support Compliance Management in Business Processes. Proceedings of the 11th Int. Conference on Electronic Commerce and Web Technologies (EC-Web 2010), Springer, 2010 (to appear).
- [STK+10] D. Schumm, O. Turetken, N. Kokash, A. Elgammal, F. Leymann, W-J. van den Heuvel: Business Process Compliance through Reusable Units of Compliant Processes. Proceedings of the 1st Workshop on Engineering SOA and the Web (ESW'10), Springer, 2010 (to appear).
- [Tre08] J. Tretmans: Model Based Testing with Labelled Transition Systems. Proceedings of the Int. Conference on Formal Methods and Testing, LNCS 4949, Springer, 2008.
- [Var01] M. Vardi: Branching vs. Linear Time: Final Showdown. Proceedings of the Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 1-22, 2001.