D2.2

Version:	2.0
Date:	2009-06-08
Dissemination status:	PU
Document reference:	D2.2

Initial Specification of Compliance Language Constructs and Operators

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	Vienna University of Technology (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

D2.2 Initial Specification of Compliance Language Constructs and Operators

Revision 2.0

Due date of deliverable: 2008-12-31

First submission date: 2008-12-30

Latest submission date: 2009-06-08

Organisation name of lead partner for this deliverable:

COMPAS Tilburg University, NL

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	2008-12-19	All sections	New document	Tilburg University
0.2	2008-12-23	All sections	Changes due to reviews: USTUTT_D2.2rev542 and RV-TUV_D2.2rev559	Tilburg University
0.3	2008-12-24	Reference	References updated due to review	Tilburg University
0.4	2008-12-30	Whole document	Minor changes and file name change du to QM review (RV-TUV_D2.2rev608)	Tilburg University
1.1	2009-05-21	All sections	Based on first year review results	Tilburg University
2.0	2009-06-08		Approval & Release	TUV

Authorisation

No.	Action	Company/Name	Date
1	Prepared	Tilburg University	2008-12-30
2	Approved	TUV	2008-12-30
3	Released	TUV	2008-12-30
2	Approved	TUV	2009-06-08
3	Released	TUV	2009-06-08

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

List of figures

List of tables

Abstract

This deliverable focuses on the introduction of initial Compliance Request Language (CRL) for the formal specification of compliance requirements that stem from: (i) Legislative and regulatory bodies (e.g.: Sarbanes-Oxley [SOX2002] and Basel II [BaselII2004]), (ii) Standards and code of practice and (iii) Business partner contracts. We have focused on the compliance requirements emerging from legislation and regulatory bodies; however the results are expected to be equally applicable to other sources of compliance. An overview of the state-of-the-art in compliance languages, particularly focusing on languages for regulatory and legislative provisions was provided in Deliverable 2.1 [D2.1]. Mandatory features that should exist in a compliance specification language were also identified. These features fall in four categories; heterogeneity (e.g. uniform specification and symmetry of specification), expressive power (e.g. conceptually natural semantics and prioritization), manageability (e.g. documentation and life cycle management) and usability (e.g. customizable and interactive). Compliance checking should be considered in two main stages of the business process life cycle, which are the design time and runtime verification and validation. While we consider the two verification stages to be mandatory and complementary, our focus in this deliverable is on the design time verification and validation

In this deliverable, we identified the languages that can be used as the basic building blocks for a comprehensive CRL. Next, we examined these languages to evaluate their feasibility in the COMPAS project. In particular, we analysed the following two families of languages. The first family of languages is the deontic logic formalism, such as Formal Contract Language (FCL) [GM05]. The second class of languages, which are used for the formal specification of compliance requirements, is built on top of temporal logic, such as Linear Temporal Logic (LTL), and Computational Tree Logic (CTL) [Var01]. We also considered a third class of compliance languages that is grounded on XML, this class is either based on deontic logic or temporal logic. For example, we considered the XML Service Request Language [APY+02], which is based on Computational Tree Logic (CTL), thus we consider XSRL to be a sub-class from the temporal logic family. In order to assess their applicability, we selected a representative language for each of these classes. Specifically, for the deontic logic formalism, we examined FCL; for the temporal logic approach, we analysed LTL; and for the XML-based approach, we assessed XML Service Request Language (XSRL).

In assessing the applicability of the approaches, we applied each representative language on the "Internet Reseller Scenario" case study introduced in Deliverable 6.1 [D6.1]. We conducted a comparative analysis between these approaches with respect to the capabilities and limitations of each representative language.

Based on the comparative analysis, the temporal logic approach was selected mainly due to the support of sophisticated automated verification tools, its proven success on the verification of a large number of complex designs and the maturity of relevant languages in the field. In addition, the approach satisfies the majority of the features identified in Deliverable 2.1 [D2.1]. Based on this detailed analysis, an initial version of the CRL is conceptualised, by defining its basic operators and compliance concepts using a meta-model. The meta-model is based on the property specification pattern system introduced in [DAC98] and extended in [YMH+06], which can be mapped to temporal logic formalisms, more specifically to Linear Temporal Logic (LTL).

1. Introduction

Today's business climate demands a high rate of compliance of business processes with which Information Technology (IT)-minded organizations are required to cope. Compliance regulations, such as Basel II [BaselII2004] , Sarbanes-Oxley [SOX2002] and others require all organizations to review their business processes and ensure that they meet the compliance standards set forth in the legislation. This includes, but is not limited to, data acquisition and archival, document management, data security, financial accounting practices, shareholder reporting functions and to know when unusual activities occur. In a broader perspective compliance can pertain to any explicitly stated rule or regulation that prescribes any aspect of an internal or cross-organizational business process; including for example public policies, customer preferences, partner agreements and jurisdictional provisions. Compliance checking should be considered in two main stages of the business process life cycle, which are the design time and runtime verification and validation. Design time verification and validation means that compliance is ensured on the design-level (static-level) of the business process models. A process model is considered as compliant with the set of relevant compliance constraints if it allows only for the execution of process instances not violating these constraints [LGR+08]. On the other hand, runtime verification and validation (dynamic verification) refers to the monitoring of the running business process instances and react probably if a violation occurs. While we consider the two verification stages to be mandatory and complementary, our focus in this deliverable is on design time verification and validation.

The main goal of work package 2 is to provide expressive languages for compliance concerns. In Deliverable 2.1 [D2.1], an overview of the state-of-the-art in compliance languages, particularly focusing on languages for regulatory and legislative provisions was presented. Current compliance-specific solutions addressing different aspects of compliance were also reviewed. In addition, a wide range of compliance legislations ([SOX2002], [BaselII2004], [FINRA08], [IFRS01]) were studied and analysed. Based on this survey and analysis, a set of core compliance concerns was identified and categorised into two groups. The first group - the basic compliance concerns- pertains to the basic structure of business processes, and relates to their workflow, information usage, location related details, employed resources and temporal occurrences. The second group -advanced concerns- is built on the basic concerns and includes monitoring, payment, privacy, quality, retention, security and transaction concerns. The relationships between basic and advanced concerns as well as interrelationships between advanced concerns were also established.

The Deliverable 2.1 also analysed existing solutions for modelling compliance concerns and investigated relevant works for each of the advanced compliance concerns. The results formed a point of departure for the development of views for the identified compliance concerns using the Model-driven Integration Architecture for compliance. A variety of compliance requirements specification languages were analysed and a set of mandatory features were identified. These features were categorised into four groups; heterogeneity, expressive power, manageability and usability. Finally, the deliverable proposed a method for modelling compliance requirements at three different levels of abstraction - goals, policies and rules –, which caters for the requirements of different stakeholders involved in compliance checking, including business analysts and compliance experts.

This deliverable represents an extension to Deliverable 2.1 [D2.1] and proposes an initial specification of the compliance language for the formal specification of compliance concerns to satisfy the set of features identified in Deliverable 2.1 and the evaluation criteria proposed in Deliverable 6.1 [D6.1]. The CRL is evaluated against the case studies proposed in Deliverable 6.1 [D6.1] by PricewaterhouseCoopers (PWC).

1.1. Purpose and scope

In the European Commission 7th framework COMPAS project on compliance-driven models, languages, and architectures for services, models, languages, and an architectural framework are being developed including required software components and services to ensure dynamic and on-going compliance of software services to business regulations (e.g. Basel II [BaselII2004], IFRS [IFRS01] and Sarbanes-Oxley [SOX2002]) and the user service requirements.

The main purpose of the report is to identify and further assess the candidate formal languages that can be utilised for the specification and representation of compliance requirements and introduce an initial specification of compliance language, along with its associated constructs and operators. The analysis starts with assessing the supported constructs and operators of existing formal languages and exploring their capabilities and limitations by applying them on a realistic example introduced in Deliverable 6.1 [D6.1]. We demonstrated and analysed the application of selected formal languages by formalizing relevant compliance requirements from the running example. We conducted a comparative analysis between candidate languages and assessed them with respect to ‘desirable features’ identified in Deliverable 2.1 [D2.1] and ‘evaluation criteria’ introduced in Deliverable 6.1 [D6.1].

The CRLs that are studied in this deliverable are closely related to Deliverable 4.1 [D4.1] and Work Package 1 [DOW] as the CRL should be integrated with the process fragments concept outlined in Deliverable 4.1 [D4.1].

1.2. Document overview

This report is organised as follows: Section 2 presents the analysis and evaluation of the candidate CRLs and their application on the case study. Section 3 introduces the three levels of abstraction - goals, policies and rules - to be used in representing compliance requirements. Also in this section, an initial CRL is introduced for representing ‘rules’. Finally conclusions and open issues are highlighted in Section 4.

1.3. Definitions and glossary

1.4. Abbreviations and acronyms

Abbreviation	**Full name**
BPEL	Business Process Execution Language
BPSL	Business Property Specification Language
CTD	Contrary To Duty
CRL	Compliance Request Language
CSP	Communicating Sequential Processes

CTL	Computational Tree Logic
DL	Defeasible Logic
FCL	Formal Contract Language
FINRA	Financial Industry Regulatory Authority
FSA	Finite State Automata
IFRS	International Financial Reporting Standards
LTL	Linear Temporal Logic
NFCL	Normal Formal Contract Language
PA	Process Algebra
PROPOLS	Property Specification Pattern Ontology language
QBE	Query-By-Example
QRE	Quantified Regular Expression
SOX	Sarbanes-Oxley Act
XML	eXtensible Markup Language
XSRL	XML Service Request Language

2. Compliance Request Languages

According to the state-of-the-art of the modelling and the formal specification of compliance requirements, languages can be classified into two main classes: The family of languages that is based on Deontic logic formalism (e.g. formal contract language) and languages grounded on temporal logic formalism (e.g. linear temporal logic, computational tree logic). We also consider a third family of languages that is built on XML (e.g. XML service request language). The XML family can be considered as a sub-class of either Deontic logic or temporal logic. For example, the XSRL (XML service request language) is an XML-based language that is grounded on Computational Tree Logic (CTL). So, we consider XSRL to be a sub-class of the temporal logic family.

To be able to explore and evaluate the strengths and limitations of these classes of languages, we studied a subset of the compliance requirements introduced by PWC in Deliverable 6.1 [D6.1] within the 'Internet reseller scenario', and used it as a running example (throughout this deliverable the term compliance requirement and compliance constraint will be used interchangeably). The scenario taking place in an online product selling system is subject to multi-compliance requirements with information processing objectives. These objectives can be organised into four categories, being completeness, accuracy, validity and restricted access. We focused on the 'Order Processing' sub-process and its relevant compliance requirements. In Table 1, each compliance constraint is described in terms of: (i) The 'ID' column gives a unique identification for the compliance constraint, (ii) the identified risk, (iii) one or more suggestive control activities to mitigate the identified risk, the control activities represent the compliance constraints that must be fulfilled, (iv) and the 'type' column typifying the compliance requirement in accordance with the four basic compliance types identified in Deliverable 2.1 [D2.1]. The four basic types identified in Deliverable 2.1 [D2.1] are:

- The ***control flow*** compliance concern encompasses requirements concerning how things are done in business processes (i.e. what activities are carried out and in what order).
- The ***information*** compliance concern deals with the information used and produced in business processes as well as the syntax and semantics of this information.
- The ***resource*** compliance concern considers the question of which resources are used within business processes (e.g. employees and customers, and computerized systems).

The ***temporal*** compliance concern takes requirements concerning when things are done/must not be done within a business process into account (e.g. in terms of relevant business events). Initially, all ten compliance requirements listed in Table 1 are validated during design time. We would like to emphasis in this study on the importance of the integration between design time validation and runtime monitoring and verification. This is due to the fact that a system can be compliant to a specific compliance requirement during design time, but might violate that requirement during runtime due to human and/or system errors or intervention. However the main focus of this deliverable is on design-time validation and verification which should be further integrated with runtime validation and verification. Hence, we consider the compliance requirements listed in Table 1 to be validated first during design time, then monitored and verified during runtime. As indicated in the 'Type' column of Table 1, the compliance requirements listed in the table encompass all four basic types of compliance concerns (flow, temporal, data and resource), and consequently considered as representative and generic.

ID	Risk	Example Control Activities (Compliance constraints)	Type
R1	Orders are not recorded completely and accurately.	The Web service automatically validates sales order data input (e.g. customer name and number, prices, terms, and credit limits) against master file data. Entries with invalid, missing or incomplete information are rejected for re-entry or stored in a suspense file for follow-up.	Data
R2		Because the customer enters sales order data via a Web service, customers are responsible for ensuring the completeness and accuracy of sales order information. Functionality is built into the Web service to allow a customer to verify the accuracy and completeness of their order, as well as the ability to edit their order before actual order submission.	Data/ Flow/ Resource
R3	Duplicate sales are recorded.	Sales orders are sequentially pre-numbered/ automatically numbered by the system. Missing or duplicate sales orders are investigated and followed-up by the sales supervisor.	Resource/ data
R4		Computer-generated sales order confirmations are sent to customers for order acknowledgement at the end of each day or on the next working day.	Data/ temporal
R5	Sales terms and prices are not approved by the appropriate level of management.	Sales orders over a set threshold require approval by management before acceptance by the system. The lack of approval creates a suspense file that is reviewed by management for clearance on a regular basis.	Resource/ data/ Flow

ID	Risk	Example Control Activities (Compliance constraints)	Type
R6		Management review and approve discounts and allowances in excess of predefined limits.	Resource/ data
R7	Sales to fictitious customers (on credit) are not prevented and detected.	Verify customer information against approved customer standing data (i.e. addresses, credit limits, etc.). Items that are not matched are researched, corrected and re-entered as necessary on a timely basis.	Data/ temporal
R8		Management review transaction files periodically for delinquent orders.	Resource/ temporal
R9	Customers' credit limits are not controlled.	Credit limits are established as part of accepting new customers. Sale orders and outstanding receivables are compared to establish credit limit before a new order is processed. Orders in excess of credit limit are stored in a suspense file to be resolved on a timely basis.	Flow/ data
R10	Duties are not adequately segregated.	Appropriate segregation of duties should be maintained. Specifically whether the credit, shipping and invoicing functions are segregated from accounts receivable, general ledger and cash functions.	Resource

Table 1 The Compliance requirements relevant to the "Internet Reseller Scenario" case study (Based on [D6.1])

2.1. Requirements

In addition to the features and characteristics identified in Deliverable 2.1 [D2.1], this deliverable introduces new features that should be contained in a CRL. These features can be summarised as follows (the features introduced in this deliverable are indicated with *):

- **Formality:** The CRL should be formal in order to allow the augmentation of business process models with compliance requirements for visualization -which in turn will help the business and compliance experts to understand the interplay between the two specifications- and pave the way for future automatic analysis, reasoning and validation techniques.
- **Expressiveness:** The CRL should be expressive enough to be able to capture the intricate semantics of compliance requirements.
- **Complexity:** The CRL should not be excessively complex to inhibit process experts to understand and use it. This characteristic can be compensated with a graphical interface mapping for the language. Expressivenesses and complexity may contradict with each other.
- **Non-determinism*:** The request languages are matched under uncertainty, respecting non-determinism of business processes. For example, the execution path to be followed can be dependent on whether the customer is 'premium' or not.
- **Consistency checks*:** Contradictions and conflicts may arise between compliance requirements particularly when they originate from different sources. It is desirable for the CRL to provide mechanisms to identify and resolve these inconsistencies and conflicts.
- **Declarative:** Compliance requirements are commonly normative and descriptive, indicating what needs to be done. Therefore, declarative languages are more suited to their formal representation as opposed to procedural languages.

- **Generic:** Compliance requirements can be constraints on the control-flow (sequence and timing of activities), data (data validation and requirements), and resource perspectives (task allocation and data access rights). The CRL should enable the representation of the requirements regarding to these perspectives.
- **Symmetricity:** This refers to the ability to model and formally specify compliance requirements and to annotate business process models with the compliance requirements. The annotation is also useful for computing a quantification of how compliant a business process is. For example, [LSG07] introduces the notion of *compliance distance* as a quantification of the efforts required to transform a non-compliant business process model to a compliant one.
- **Monotonicity:** A violation to a compliance rule is not necessarily an error. In some circumstances, it is desirable to tag certain compliance rules as monotonic and others as non-monotonic. Monotonic rules are the rules that cannot be violated from a business point of view; whilst, non-monotonic rules are open to violation to a certain extend and under specific conditions. Depending on the rigidity of the rule, the process expert determines the type of the rule. For the definition of non-monotonic rules, the CRL can provide mechanisms to enable process experts to prioritise the rules.

2.2. Current Approaches

This section provides a description of the two major classes of languages utilised for modelling compliance requirements and analyse representative languages. First, the languages based on deontic logic formalism (such as the ones introduced in [GMS06], [GM05], [LSG07], [GV06] and [SGN07]) are discussed. Second, the languages established on temporal logic formalism (such as the ones utilised in [GLM+05], [AM07], and [LMX07]) are examined. Finally, an XML-based approach -XSRL introduced in [APY+02] and extended in [LAP06]- is discussed.

To study the applicability of each approach with respect to the requirements set forth in section 2.1 and to assess the efficiency of the associated verification approaches, we selected the most recent proposal from each class and tried to formally specify the compliance requirements listed in Table 1 of the "Internet Reseller" scenario. More specifically, for the deontic logic formalism, we selected the Formal Contract Language (FCL) introduced in [GM05] and reused in [SGN07]. For the temporal logic formalism, we examined the Linear Temporal Logic (LTL) mainly due to its expressiveness, intuitiveness and its support for compositional and semi-formal verification. Finally, for the XML-based approach we evaluated XML Service Request Language (XSRL) [APY+02], [LAP06], which is considered as a sub-class of the temporal logic family as it is grounded on Computational Tree Logic (CTL).

2.2.1. Deontic Logic Approach

Authors in [GM05] provided the foundations for the Formal Contract Language (FCL). They considered the compliance requirements that stem from business contracts. However, as proposed in [SGN07], FCL can also be used to express other types of compliance requirements emerging from legislation and regulatory bodies, and standards and code of practice. FCL is a combination of an efficient non-monotonic formalism (defeasible logic) and deontic logic of violations achieving the right balance between expressiveness and computational complexity. As stated in [GM05], defeasible logic (DL) proved to be a flexible non-monotonic formalism able to capture different and sometimes incompatible facets of non-monotonic reasoning. The primary usage of DL in FCL is the identification and resolution of conflicts that may arise between different compliance rules. More specifically, according to DL, If there is a state where we can conclude both A and ¬A, then neither of them will be concluded, unless a superiority relation is defined between them. For example, if the designer specifies that A is superior over ¬A, then A will be concluded. In addition to the conflicts arising between a proposition and its negation, the designer can define the set of incompatible literals, which are the ones that can't hold at the same time. For example, a customer can't be a premium customer and a basic customer at the same time. Consequently, the designer should specify that {PremiumCustomer, BasicCustomer} are incompatible literals meaning that PremiumCustomer(a) and BasicCustomer(a) for a Customer 'a' can't hold at the same time. The *superiority relation* in defeasible logic is used to specify a prioritization between rules, where a rule with a higher priority can override the conclusion of another rule with lower priority.

In deontic logic, compliance requirements should be reduced to the set of obligations, prohibitions and permissions (and other normative positions) the enterprise has to follow in order to be considered as compliant. Deontic logic of violations provides the ability to reason about violations, and the obligations arising in response to violation. Reparation to a specific violation can take the form of a reparation chain.

The basic mechanism of the logic of violation takes a modular approach to the problem. This modularity is of particular importance to the compliance, since the nature of today's enterprises is based on the composition of diverse components. Thus, it is possible to review a specification of a component of a business process or specific subset of compliance rules without the need to make a comprehensive revision of the representation of the business process or the entire set of compliance rules.

FCL is based on a constructive approach for proofs, where the set of derivations that leads to a particular conclusion can be obtained. This is significant in the sense that it is not only desirable to know that a business process doesn't comply, but it is also important to know why it doesn't comply to be able to resolve the non-compliance.

The FCL language consists of two sets of atomic symbols: A finite set of literals (propositions) that represent state variables, and a finite set of events. The logical operators that are supported are as follows:

(i) ";" the sequence operator

(ii) "Λ" conjunction operator

(iii) "v" disjunction operator

A rule in FCL is an expression of the form r: $A1, A2, \ldots, An \vdash B$, where 'r' is the identification of the rule, A1, A2, … An is the set of premises (propositions) and B is the conclusion of the rule. The rule is built from a finite set of atomic propositions, logical operator, and a set of deontic operators, which are; (i) Negation (¬), (ii) Obligation (O), (iii) Permission (P), and (iv) Contrary to duty ($\otimes$). Contrary to duty (CTD) operator is used to specify the violations and the obligations arise as a response to the violations. The rules are formed as follows:

- Each atomic proposition is a proposition.
- If P is an atomic proposition then $\neg P$ is a proposition.

- If P is a proposition then O_P is an obligation proposition and P_P is a permission proposition. Obligation proposition and permission proposition are deontic propositions.
- Prohibitions can be either represented as O¬ or ¬ P.
- If $P_1, P_{2,} ..., P_n$ are obligation propositions and q is a deontic proposition then

  $P_1 \otimes P_2, \otimes ... \otimes P_n \otimes q$ is a reparational chain.

The reperational chain indicates that, if P_1 is violated, its violation can be repaired by the secondary obligation P_2. If, P_2 cannot be satisfied then it can be repaired by obligation P_3, and the chain continues. Usually the permission appears as the last deontic proposition in a reparation chain, which is intuitive from a business perspective. Rules are classified into:

(i) Definitional rules; which corresponds to factual statements, and,

(ii) Normative rules; which allows us to conclude normative positions.

Temporal dimension is an important aspect that must be incorporated into definitions. To incorporate the temporal dimension, all propositions can be time-stamped and a persistence approach can be adapted as proposed in [GRS05]. According to the persistence approach, if we can conclude *P* at time *t,* -written as *P*:*t* -, then *P* is true for all *t'* > t, until an event occurs that terminates the validity of *P*.

According to the deontic logic of violations, a normative document consists of a set of normative clauses and these clauses cannot be considered in isolation, i.e., the entire document should be conceived as a whole. This is one of the reasons for the transformation of a FCL representation to its normal form (NFCL). The normalization process is performed to clean up the FCL representation; to identify loopholes, deadlocks, inconsistencies, conflicts and most importantly to make hidden rules explicit. The process consists of two steps; first, merging the rules to bring related ones together to explicitly represent reparation obligations; and second, removing the redundancies and identifying the conflicts and resolving them.

According to [GMS06], in order to check the compliance, business process models should be reduced to a possible set of execution paths. An execution path is the trace of execution tasks in a process model. To be able to check the compliance between FCL specifications and business specifications, two specifications should be brought to the same level of abstraction. To enable this, business process models are transformed to an event–based language since the FCL language is event-oriented. Compliance can be checked between the two specifications following the notion of *Idealness* (ideal, sub-ideal, non-ideal and irrelevant) introduced in [GMS06]. An *ideal situation* is a situation where execution paths do not violate FCL expressions, and thus the process model is fully compliant with the FCL rule. In a *sub-ideal situation,* there exists violations, but they are repaired by reparation actions. In a *non-ideal* situation, violations occur without reparation. Finally, no rule is applicable to the *irrelevant* situation. Here the situation represents a specific execution path in a business process model.

A business process representation is 'ideal / sub-ideal / non-ideal / irrelevant' with respect to a FCL specification if the business process representation is 'ideal / sub-ideal / non-ideal / irrelevant with respect to its normal form. Since the reverse is not true, the normal form - not the FCL specification itself- should be checked for compliance.

To integrate design time validation with runtime monitoring and consequently to provide lifetime compliance support, FCL can be further mapped to RuleML [RuleML05] by integrating the work performed in [GM05]. RuleML is a generic extensible and semantically neutral rule markup language mainly utilised for exchanging rules. RuleML programs are not intended to be executed directly. However, for execution, the business logic of RuleML programs can be implemented via XSLT transformations into the target language of the receiving rule-based systems.

Formal representation of the requirements using FCL

In the following paragraphs, we specify the formal representation of the requirements presented in Table 1 using FCL.

For R1

r1.1: customer(x), salesOrder(y,x):t, masterFile(z, x):t $\vdash O_{sys}$ validateOrder(y,z):t

r1.2: validateOrder(y,z):t, InvalidEntries:t, t'>t $\vdash O_{sys}$ reject(y):t' $\otimes$

O_{sys} StoreSuspenseFile (y):t'

salesOrder(x,y):t predicate represents a customer 'x' who has created a sales order 'y' at time t. The predicate masterFile(z, x):t represents the master data file 'z' containing customers' information at time t. InvalidEntries:t predicate represents the invalid entries at time t. reject(y):t' predicate indicates that at time t', where t'>t, it is an obligation on the system to reject the sales order 'y'. If this obligation cannot be satisfied, then the predicate StoreSuspenseFile (y):t' should take place, implying that the sales order 'y' should be stored in a suspense file at time t'.

For R2:

r2.1: customer(x):t, EnterOrder(l, x):t, t'>t $\vdash O_{C}$ reviewEdit(l):t'

r2.2: customer(x):t, reviewEdit(l, x):t', t''>t' $\vdash O_{C}$ submitOrder(l):t''

The predicate EnterOrder(l, x):t represents the customer 'x' enters the order 'l' at time t. reviewEdit(l):t' indicates that at time t' (where t'>t), the customer 'x' should review and edit the order 'l'. At time t'' (where t''> t'), the customer 'x' submits the order 'l' (submitOrder(l): t'').

For R3:

r3.1: Timer(t) $\vdash O_{SS}$ Review(sales_orders):t

r3.2: Timer(t) $\vdash O_{zyx}$ UpdateTimer(t, k):t r3.3:Review(sales_orders):t, MissingDuplicate(wrongSalesOrders):t $\vdash$ t'>t O_{SS} Investigate(wrongSalesOrders):t'

The predicate Timer(t) acts as a timer trigger when the current time is t. The predicate UpdateTimer(t, k):t indicates that at time t the timer is updated by k time units. This predicate captures the concept of "on a regular basis". The other predicates are similar to the ones introduced in prior definitions and are self-exploratory.

For R4:

r4.1:customer(x):t, salesOrder(y,x):t $\vdash$ O_{zyx} SendCustomerConf(c,x):t+k $\otimes$ O_{zyx} SendCustomerConf(c,x):t+ 24

The predicate salesOrder(y,x):t indicates that at time t, there is a sales order 'y' related to the customer 'x'. The company 'zyx' is obliged to send the customer 'x' a confirmation within k hours from receiving the sales order. If not satisfied, the 'zyx' is obliged to send the confirmation within 24 hours after receiving the sales order.

For R5:

r5.1: customer(x):t, salesOrder(y,x):t, price(y, threshold):t $\vdash$ O_{manger} Approve(y):t+k $\otimes$ StoreSuspenseFile (y):t+k

r5.2: Timer(t') $\vdash$ O_{manger} clearSuspenseFile:t'

r5.3: Timer(t') $\vdash$ O_{zyx} UpdateTime(t', k'):t'

The predicate price(y, threshold):t indicates that the price of the sales order 'y' is above a given threshold value at time t. Timer predicate has the same meaning as illustrated in r3.

For R6:

Customer(x):t, salesOrder(y,x):t, discount(y, dis_limit):t $\vdash$ O_{manger} Approve(dis_limit):t+k

The predicate discount(y, dis_limit):t denotes that the discount on the sales order 'y' is above a certain limit.

For R7:

r7.1: customerInfo(x):t, cusStandData(st,x):t $\vdash$ O_{zyx} verifyCustomerInfo(x, st):t r7.2: verifyCustomerInfo(x, st):t, ¬ matchedItems(i):t $\vdash O_{zyx}$ researchCorrect(i):t+k

The predicate cusStandData(st,x):t represents the customer standing data ‘st’ for Customer ‘x’ at time t. The predicate matchedItems(i):t represents matched items ‘i’ at time t by verifying the customer info ‘x’ against customer standing data ‘st’.

For R8:

r8.1: Time(t) $\vdash O_{manger}$ ReviewTransactionFiles:t;

r8.2: Time(t) $\vdash O_{zyx}$ UpdateTime(t, k):t

For R9:

r9.1: SalesOrder(l, x):t , outRecieveable(c, x): t $\vdash O_{zyx}$ obtainCreditLimit (x):t

r9.2: obtainCreditLimit(x):t $\vdash O_{zyx}$ AcceptCustomer(x):t

r9.3: obtainCreditLimit(x):t, EnoughLimit:t, t’ $\vdash$> t O_{zyx} ProcessOrder(l): t’ $\otimes$ StoreSuspenseFile(l): t+k

r9.4: obtainCreditLimit(l, c):t, ¬EnoughLimit:t $\vdash O_{zyx}$ StoreSuspenseFile(l): t+k

r9.5: suspenseFile(y):t $\vdash O_{actor}$ resolve(y): t+k

For R10:

r10.1 : Officer(y):t, Customer(x):t; CheckCredit(x,y):t , officer(z): t’, z ≠ y: t’ $\vdash$

O_{zyx} receiveCash(x,z):t’

r10.2 : Officer(y):t, Customer(x):t; createShipOrder(x,m,y):t , officer(z): t’, z≠ y: t’ $\vdash O_{zyx}$ receiveCash(x,z):t’

r10.3 : Officer(y):t, Customer(x):t; createInvoice(x,i,y):t , officer(z): t’, ≠ y: t’ $\vdash O_{zyx}$ receiveCash(x,z):t’

OR

r10.1' : Officer(y):t, Customer(x):t; CheckCredit(x,y):t , officer(z): t', receiveCash(x,z):t', z = y: t' $\vdash O_{zyx}$ AlarmManager:t'

r10.2' : Officer(y):t, Customer(x):t; createShipOrder(x,m,y):t , officer(z): t', receiveCash(x,z):t', z = y: t' $\vdash O_{zyx}$ AlarmManager:t'

r10.3' : Officer(y):t, Customer(x):t; createInvoice(x,i,y):t , officer(z): t', receiveCash(x,z):t', z = y: t' $\vdash O_{zyx}$ AlarmManager:t'

The requirement r10 requires further interpretation because of the ambiguities regarding the duties of 'accounts receivable' and 'general ledger' units. When we assume that r10 requires segregation of credit, shipping and invoicing functionalities from cash functionality, two discrete alternatives emerge. We can follow a preventive approach as in rules r10.1, r10.2, and r10.3 (recommended); or a retrospective approach as in rules r10.1', r10.2' and r10.3'. Here, the predicate CheckCredit (x,y):t indicates that at time t, officer 'y' has checked the credit worthiness of customer 'x'. The predicate createShipOrder(x,m,y):t indicates the shipping order 'm' for customer 'x' created by officer 'y' at time 't'. The meaning of createInvoice is similar. Officer(y):t means that at time t, 'y' plays the role of the officer.

2.2.2. Temporal Logic Approach

This direction exploits the sophisticated model-checking tools associated with process algebra (PA) for the automatic verification of a system specified using PA formalisms such as Communicating Sequential Processes (CSP), Π-Calculus and Timed CSP, against some properties specified using temporal logic formalisms such as Linear Temporal Logic (LTL) and Computational Tree Logic (CTL). Of special interest is the work recently performed in [LMX07] where LTL is used as the CRL. As mentioned in [LMX07], LTL was chosen over CTL since the branching-time formalisms of CTL is unintuitive to business analyst and doesn't support compositional reasoning (refer to section 3 for a detailed discussion).

LTL is a logic used to formally specifying temporal properties of software or hardware designs. In LTL each state has one possible future and can be represented using linear state sequences, which corresponds to describing the behaviour of a single execution of a system.

The formulas in LTL take the form $A\ f$, where A is a universal path quantifier and f is a path formula. A path formula must contain only atomic propositions as its state sub-formulas.

The formation rules for LTL formulas are as follows:

- If $P \in AP$, where AP is a non-empty set of atomic propositions, then P is a path formula.
- If f and g are path formulas, then $\neg f$, $f \vee g$, $f \wedge g$, $\mathbf{X} f$, $\mathbf{F} f$, $\mathbf{G} f$, $f\,\mathbf{U} g$ are path formulas (where '$\vee$'represents 'or', '$\wedge$' represents 'and' operators), such that:
  - **X (**next time) indicates that the formula f is true in the second state of the path.
  - **F (**eventually) indicates that formula f will be true at some state in the future of the path.

- **G (**always) indicates that formula f must be true in all the states of the path.
- **U (**until) indicates that if at some state in the future the second formula g will be true, then, the formula f must be true in all the subsequent states within the path.

As opposed to the case for deontic logic formalism, in LTL, the role that is expected to perform a certain action or operation cannot be annotated to the operators. However, roles can be annotated to actions in LTL as one of the parameters of the action. For example, validateOrder(x,y) action performed by 'z' can be represented as 'Action = validateOrder & paralist = {x, y, z}' in LTL.

In [LMX07], a set of model transformations is performed in order to check the compliance. The study assumes that business processes are modelled using Business Process Execution Language (BPEL), which are then transformed first to Π-Calculus and then from Π -calculus to finite state machine representations in order to bring the compliance and business specifications in syntax acceptable for the model-checker - NuSMV2. The model-checker automatically checks the compliance between the two specifications. If the system is not compliant with one of the compliance rules, the model-checker provides the *counterexample tracing* facility that helps the process expert to resolve non-compliance. The counterexample tracing facility identifies the fragments of the business process model that are the source of non-compliance and consequently allowing the process experts to just focus on these fragments for the non-compliance resolution.

One of the main problems of this approach is the complexity of the LTL (and similar formalisms) to be used and understood by compliance and business experts, who are usually considered not to possess a strong mathematical background. The study [YMH+99] proposes a language PROPOLS (Property Specification Pattern Ontology language), which is based on Dwyer's property patterns introduced in [DAC98]. A *property pattern* is a high level abstraction of frequently used temporal logic formula. Each property specification pattern has (i) a pattern; defining what must occur, and (ii) a scope; indicating when the pattern should occur. The use of the property pattern helps the compliance and business experts to define and understand formal specifications and consequently helps bridging the gap between compliance and business experts from one side and model checkers from the other side.

The property pattern system introduced in [DAC98] is extended in [YMH+99] to enable the nesting of patterns, and consequently to be able to recognise complex compliance requirements. The study uses the logical operators (Not, And, Or, Xor, If then) for the composition of Dwyer's property patterns. The semantics of the composite property patterns are represented by Finite State Automata (FSA). The study also proves the correctness of the compositions. In this respect, PROPOLS language is a suitable candidate for the formal specification of compliance requirements. Automatic mapping can be performed from PROPOLS to LTL (or CTL) as proposed by [DAC98]. Next, model-checker tools can be used to validate business process models against compliance requirements during design time.

For runtime monitoring, the work performed by [NS07] can be integrated with [YMH+99]. [NS07] proposes a model-driven approach for internal controls of compliance in business processes. The study also makes use of Dwyer's property specification patterns for the formal specifications of the compliance requirements. The approach consists of three phases. The first phase, called "Semantic process mirror", is an abstract layer on the top of the syntactical business process description. In this phase a semantic model is developed. This semantic model links business process's activities along with their agents and business documents to a set of internal controls responsible for ensuring the compliance. The set of recovery actions that should be automatically executed as a response to controls violations is also represented in this semantic model. The second phase is the application control design phase, where compliance requirements are formalised based on Dwyer's property specification patterns. The last phase is the business process execution phase, which allows the bidirectional interaction between business process and compliance requirements. In this phase, "Semantic process mirror" is updated by the information about the current instance of the business process. If a violation occurs, the appropriate recovery action defined in the "Semantic process mirror" fires.

To be able to capture complex compliance requirements, Dwyer's property specification patterns exploited in [NS07] for runtime monitoring should be extended by the notion of *composite patterns* introduced in [YMH+99]. However, the transformation between PROPLOS language and the logical formalisms proposed in [NS07] requires further investigation and can be considered as a subject in deliverable 2.3 [DOW].

Formal representation of the requirements using LTL

In the following paragraphs, we specify the formal representation of the requirements presented in Table 1 using LTL (role assignments are provided only for actions where it is deemed necessary).

For R1:

G ((Action = SalesOrder & paralist = {y, x}) ∧ (Action = masterFile & paralist = {z, x}) → **F** (Action = ValidateOrder & paralist = {y, z} → **X** (Action = InvalidEntries → **X** ((Action = RejectOrder & paralist = {y}) ∨ (Action = StoreSuspenseFile & paralist = {y})))))

This rule indicates that when there is a 'salesOrder' and a 'masterFile', then eventually 'ValidateOrder' action should be performed. In the next state, if 'InvalidEntries' exist, then eventually either 'RejectOrder' or 'StoreSuspenseFile' actions should take place. The meanings of the parameters' are as follows: 'y' is an identifier for the sales order, 'x' is the customer, 'z' is an identifier for the masterFile.

For R2:

G (Action = EnterSalesOrder & paralist = {l, x} → **F** (Action = reviewEdit & paralist = {l} → **F**(Action = submitOrder & paralist = {l})))

This rule indicates that when the action 'EnterSalesOrder' takes place, eventually, 'reviewEdit' and 'submitOrder' actions should take place sequentially. The meanings of the parameters' are as follows: 'l' is an identifier for the sales order and 'x' refers to the customer that enters the sales order.

For R3:

G (Action = Review & paralist = {Sales_orders, SS} → **F**(Action = MissingDuplicate & paralist = {wrongSalesOrders} → **X** (Action = Investigate & paralist= {wrongSalesOrders, SS})))

This rule indicates that when 'Review' action takes place by the 'sales supervisor (SS)' for reviewing the 'Sales_orders', then eventually, 'Investigate' action should be performed by the 'sales supervisior' provided that 'MissingDuplicate' evaluates to true; where 'MissingDuplicate' serves as a flag for the existence of missing duplicate sales orders.

For R4:

G (Action = SalesOrder & paralist = {y, x} → F ((Action = SendCustomerConf & paralist = {c, x, k}) ∨ (Action = SendCustomerConf & paralist = {c, x, 24})))

This rule denotes that if a 'SalesOrder' exists, a confirmation should be sent to the customer. According to the compliance requirement description given in Table 1, the first priority is given for sending the confirmation in the same day of the date of order. If this is not achieved, confirmation is sent in the next working day. In LTL, the closest operator that can capture a part of this information is ' ∨ ' (Or).

For R5:

R5.1: **G** ((Action = SalesOrder & paralist = {y, x}) ∧ (Action = price & paralist = {y, threshold}) → **X** ((Action = Approve & paralist = {y, manager}) ∨ (Action = StoreSuspenseFile & paralist = {y})))

R5.1: **G**(Action = Timer & paralist = {t} → **F**(Action = clearSuspenseFile ∧ Action = updateTimer & paralist = {t, k}))

The problem discussed in the previous rule also holds for this rule. The ' ∨ ' operator is not capable of capturing the semantics of the CTD *(⊗)* operator used in deontic logic.

For R6:

G ((Action = SalesOrder & paralist = {y, x}) ∧ (Action = Discount & paralist = {y, dis_limit}) → **X** (Action = Approve & paralist = {dis_limit, manager}))

In this rule, specific role (manager) that performs the 'Approve' action is also represented by including it in the parameter list of the action, since it is necessary to specify and ensure that the 'manager' is responsible for the approval.

For R7:

G ((Action = verifyCustomerInfo & paralist = {x, st} → **F** (! Action = matchedItems & paralist {i} → **X** (Action = researchCorrect & paralist = {i})))

The rule given in Table 1 specifies that unmatched items must be researched, corrected and re-entered 'on a timely basis'. We interpret this statement by using the temporal operator (**X**), indicating 'Next state'. As an alternative, (**F**) operator can be used instead of (**X**) and max time units can be included as a parameter for the 'researchCorrect' proposition.

For R8:

G (Action = Timer & paralist = {t} → **X** (Action = ReviewTransactionFiles & paralist = {manager} ∧ Action = updateTimer & paralist = {t, k}))

For R9:

R9.1: **G** ((Action = salesOrder & paralist = {l,x}) $\wedge$ (Action = outRecievable & paralist = {c, x}) $\rightarrow$ **F** (Action = ObtainCreditLimit & paralist = {x}))

R9.2: **G**(!(Action = AcceptCustomer & paralist = {x}) $\vee$ (!(Action = AcceptCustomer & paralist = {x}) **U** (Action = ObtainCreditLimit & paralist = {x})))

R9.3: **G**(!(Action = ProcessOrder & paralist = {l, x}) $\vee$ (!(Action = ProcessOrder paralist = {l, x}) **U** (Action = obtainCreditLimit &paralist= {x})))

R9.4: **G**(Action = salesOrder(y,x) $\wedge$ **!**(Action = EnoughLimit) $\rightarrow$ **F** (Action = storeSuspenseFile & paralist = {y}))

R9.5: **G**(Action = suspenseFile& paralist = {y} $\rightarrow$ **X** (Action = resolve & paralist = {y, actor}))

This can be considered as a typical temporal constraint, which is successfully represented using LTL. Similarly, the statement 'on a timely basis statement' is represented by the temporal operator (**X**) – indicating 'Next state'. The alternative discussed in previous rule is also applicable here (replacing (**X**) by (**F**) and adding a 'max time units' as a parameter for 'resolve' proposition).

For R10:

R10.1: **G**(Action = credit& paralist = {x} $\rightarrow$ **G** (! Action = cash & paralist = {x}))

R10.2: **G**(Action = shipping& paralist = {x} $\rightarrow$ **G** (! Action = cash & paralist = {x}))

R10.3: **G**(Action = invoicing & paralist = {x} $\rightarrow$ **G** (! Action = cash & paralist = {x}))

This compliance requirement is considered as abstract and proper formalization of the rule requires further interpretation by the experts. With the information presented, an abstract formal specification in LTL is given in R10.1, R10.2 and R10.3 above.

2.2.2.i. XML Service Request Language (XSRL)

XSRL is declarative and formally-based on eXtensible Markup Language (XML) and Computational Tree Logic (CTL), so we consider it to be a sub-class of the temporal logic family. [APY+02] introduces the XSRL language as a service request language for XML-based web services. XSRL contains a set of rich constructs for expressing users' goals and constraints, and defining service scheduling preference and alternative options. [LAP06] proposes a planning framework, where the user is equipped by the XSRL language to express his/her request. The framework is based on the principle of interleaving planning and execution to deal with aspects such as, non-determinism, potential absence of information, and continuously changing needs and demands of users and businesses, which can only be dealt with during runtime. Therefore, in such a framework, information about the outcome of an action invocation is gathered during runtime and used to re-plan consistently to the original goal, while interacting with the user for confirmation and/or verification. Although XSRL is effective for runtime monitoring, there are no studies done on the design time validation.

In assessing the candidacy of XSRL for CRL, we adhere to the extended version of XSRL introduced in [LAP06] to deal with the interleaving between planning and execution.

Propositions are the atomic objects of the XSRL and can either be 'true' or 'false' at a certain state. The syntax of the XSRL is as follows:

- **xsrl :** *<XSRL>* goal *</XSRL>*
- **goal:** ***achieve-all*** | ***proposition*** | ***then*** | ***vital*** | ***prefer*** | ***optional*** | ***atomic*** | ***vital-maint*** | ***optional-maint***

Propositions can be combined with each other by ***sequencing operators*** described below:

- **achieve-all:** *<Achieve-All>* +goal *</Achieve-All>*

Achieve-all is evaluated to true iff all its sub-goals are evaluated to true; otherwise it is evaluated to false

- **then:** *<Before>* goal1 *</Before>*
  <then> goal2 *</then>*

Before-then is evaluated to true iff 'goal1' is evaluated to true, and from this state, 'goal2' must be evaluated to true, otherwise, it is evaluated to false.

- **Prefer:** *<Prefer>* goal1 *</prefer>*
  <to> goal2 *</to>*

Prefer-to is evaluated to true iff it is possible to satisfy 'goal1'; if this is not possible, then 'goal2' must be satisfied, otherwise it is evaluated to false. Consequently, prefer-to provides the user with the ability to prioritise goals. By nesting prefer-to clauses, a user can also provide a prioritization over any numbers of sub-goals.

XSRL also contains a set of operators that provide the user with the capability to specify *how to satisfy propositions*, where propositions are the arguments of these operators.

- **Vital:** *<vital>* proposition *</vital>*

Vital is true, if there is a state where proposition is satisfied, such that this state is reachable from any future state, otherwise, it is false.

- **Optional:** *<optional>* proposition *</optional>*

Optional indicates that, if there exists a state in the future such that proposition is satisfied, then this state must be reached, otherwise the proposition is ignored.

- **Atomic:** *<atomic>* proposition *</atomic>*

Atomic indicates that proposition must be reached from the current state despite the non-determinism of the domain, if this path does not exist, then it is evaluated to false.

- **Vital-maint:** *<vital-maint>* proposition *</vital-maint>*

Vital-maint is true if proposition is true in all the states in the execution path, otherwise it is false (vital-maint is analogous to 'always' in LTL).

- **Optional-maint:** *<optional-maint>* proposition *</optional-maint>*

Optional-maint is analogous to vital-maint, but if such a path doesn't exist, it doesn't fail.

The formation rules of propositions are as follows:

- **Proposition:** <***CONST*** ATT = "true | false"> | ***var*** |
  <And> +proposition *</And>*

  <Or> +proposition *</Or>*

  <Not> proposition *</Not>*

  <Greater> var *</Greater>* *<Than>* rval *</Than>*

  <Less> var *</Less>* *<Than>* rval *</Than>*

  <Equal> var rval *</Equal>*

Var:: a..zA….Z[rval]

Rval: +a...zA...Z0...9

Formal representation of the requirements using XSRL

In the following paragraphs, we specify the formal representation of the requirements presented in Table 1 using XSRL. For simplicity, we omitted the XML tags from the specifications (the parameters used in the specifications below hold the same meaning as the parameters used in the specifications with FCL and LTL formalisms).

For R1

```
Achieve-all
    Before
        Vital salesOrder(x, y) Λ masterFile(z, x)
    Then
        Before
            Atomic ValidateOrder(y, z)
        Then
            Before
                Atomic InvalidEntries
            Then
                Prefer vital RejectOrder(y) to
                        Vital StoreSuspenseFile(y)
```

We should note that, in this rule, it is possible to specify a prioritization between actions using the 'prefer' operator. It carries the same semantic meaning of the CTD operator of the deontic logic.

For R2

```
Achieve-all
    Before
        Vital EnterSalesOrder(l,x)
    Then
        Before
            Atomic reviewEdit(l)
        Then
            Vital submitOrder(l)
```

For R3

```
Achieve-all
    Before
        Atomic Review(sales_order)
    Then
        Before
            Atomic missingDuplicate(wrongSalesOrders) Λ
                    Equal Timer t
        Then
            Atomic Investigate(wrongSalesOrders) Λ
```

```
                    Atomic UpdateTimer(t, k)
```

The ‘Equal’ operator in this rule compares the equality of a variable to a certain numeric value. We can compare numbers using ‘Equal’, ‘Greater’ or ‘Less’ operators supported by XSRL. (In LTL or FCL this was achieved by adding the variable along with the value to be compared with as parameters in a predicate. The numerical equality operators are not supported in LTL or FCL. Consequently, from this point of view the XSRL can be considered as richer than LTL and FCL.)

For R4

```
Achieve-all
    Before
        Vital SalesOrder(y,x)
    Then
        Prefer Vital sendCustomerConf(c, x) Λ
               Less Time than k To
                 Vital sendCustomerConf(c, x,) Λ
                 Less Time than 24
```

Similar to the situation we have in r1, the ‘prefer’ operator is able to capture the semantics of the CTD operator in deontic logic. We should also highlight the usage of the ‘Less-Than’ operator in this rule.

For R5

```
Achieve-all
    Before
        Vital SalesOrder(y,x) Λ
            Greater price than threshold
    Then
        Before
            Prefer Atomic Approve(y, manager) To
                   Atomic StoreSuspenseFile(y)
        Then
            Optional  ClearSuspenseFile(manager)
```

For r6

```
Achieve-all
    Before
        Vital SalesOrder(y,x) Λ
            Greater Discount than dis_limit
    Then
        Atomic Approve(dis_limit, manager)
```

For R7

```
Achieve-all
    Before
        Atomic verifyCustomerInfo(x, st)
    Then
        Before
```

```
                **Vital** ¬ matchedItems
        **Then**
                **Atomic** researchCorrect(i)
```

For R8

```
Achieve-all
    Before
        Vital Equal Timer t
    Then
        Atomic ReviewTransactionFiles(manager) Λ
       Atomic UpdateTimer(t, k)
```

For R9

R9.1:

```
Achieve-all
    Before
        Vital Sales_order(l, x) Λ outRecievable (c, x)
    Then
        Atomic  ObtainCreditLimit(x)
```

R9.2:

```
Achieve-all
    Before
        Atomic ObtainCreditLimit(x)
    Then
        Atomic AcceptCustomer(x)
```

R9.3:

```
Achieve-all
    Before
        Atomic ObtainCreditLimit(x)
    Then
        Atomic ProcessOrder(l,x)
```

R9.4:

```
Achieve-all
    Before
        Vital salesOrder(y,x) Λ ¬ EnoughLimit
    Then
        Before
            Vital StoreSuspenseFile(y)
        Then
            Atomic Resolve(y, actor) Λ
```

We should note that, unlike the case in LTL, in XSRL, it is easier to represent that a certain action/proposition must precede another action/proposition (for the comparison, refer to the LTL specification on page 21).

For R10

R10.1:

Achieve-all

```
        Before
                Atomic credit(x)
        Then
                Atomic ¬ cash(x)
R10.2:
Achieve-all
        Before
                Atomic shipping(x)
        Then
                Atomic ¬ cash(x)
R10.1:
Achieve-all
        Before
                Atomic invoicing(x)
        Then
                Atomic ¬ cash(x)
```

2.3. Analysis and Assessment

In the previous section, we formalised the compliance requirements of the order-processing sub-process of the "Internet Reseller" case study using the representative languages of the two main approaches. This section presents the evaluation of each of these approaches and highlights their strengths and limitations.

For the evaluation purposes, the following three questions should also be considered in accordance with Deliverable 6.1 [D6.1]:

- **Q1:** Can all compliance requirements identified in WP6 case studies be expressed in the CRL?
- **Q2:** Can all expressed compliance requirements be mapped to transformation templates for generating executable variants from them?
- **Q3:** Can all compliance requirements identified in WP6 be addressed by design and/or runtime components for verification of compliance?

Table 2 summarises the comparative analysis for the two approaches in terms of, (i) the degree of support provided to the key characteristics identified in section 2.1, (ii) the operators they support, and (iii) the extent of the response for the questions cited above.

	Deontic Logic	Temporal Logic	XSRL
1. **Formality**	+	+	+
2. **Complexity**	±	±	±
3. **Expressiveness**	+	±	+
4. **Declarative**	+	+	+
5. **Non-determinism**	+	+	+
6. **Consistency Checks**	+	-	-
7. **Non-Monotonicity**	+	-	+
8. **Generic**	+	+	+

	Deontic Logic	Temporal Logic	XSRL
9. Symmetric request	±	-	-
10. Intelligible feedback	-	+	-
11. Q1	+	+	+
12. Q2	±	±	*not applicable*
13. Q3	±	+	±
14. Operators supported	• **;, Λ**, ν. • ¬, **O**, **P**, **O**¬, ¬**P** , ⊗	¬, ∨, ∧, **X**, **F**, **G**, **U**	• **Sequence**: achieve-all, then, prefer. • **Modality**: Vital, Optional, Atomic, Vital-maint, Optional-maint • **Logical:** ¬, **Λ**, ν, >, < , =

Table 2 A comparative analyses of compliance specification approaches

The first nine rows in Table 2 represent the identified features and characteristics summarised in section 2.1. The tenth row is an additional criteria introduced to represent whether the approach provides the user with meaningful feedback in cases of non-compliances. Rows 11 to 13 represent the three evaluation questions. The degree of support is denoted by; (i) '+', indicating that the feature is satisfied, (ii) '-', indicating that the feature is not satisfied, and (iii) '±', indicating that the support is partial. The last row lists the main operators supported by each approach.

2.3.1. Evaluation of Deontic Logic Formalism

As shown in Table 2, deontic logic formalisms, more specifically the FCL, is expressive enough to capture the semantics of compliance rules relevant to the running example used in this deliverable. Using FCL, we were able to represent all of the compliance requirements presented in Table 1. The FCL language is declarative and it supports non-determinism. For example; a case such that a state 'st' moves to a next state 'st1' or 'st2' under certain condition(s), can be represented by two FCL rules. The first rule represents the case where 'st' moves to 'st1' (and the relevant condition(s) can be represented in the antecedent part of the rule), and the second rule represents the other case where 'st' moves to 'st2'.

FCL provides a mechanism for consistency checks by the means of the superiority relation of the deontic logic. Non-monotonic requirements can also be met by means of the superiority relation, where a prioritization can be provided between the set of related non-monotonic rules. FCL was generic enough to represent different types of compliance requirements (sequence, temporal, data and resource perspective) relevant to the running example. By exploiting the work performed in [SGN07], which uses FCL as the CRL, business process models can be visually annotated by compliance requirements using the *control tags*. The 'symmetric request' feature is partially supported by FCL, because, as clarified in Section 2, 'symmetric request' feature requires actual augmentation of business process models by compliance rules (as opposed to having only visual annotations).

For FCL, the answer to Q1 is “Yes”; however, this response should be further analysed by including refined case studies from other industry partners. This also holds in spite of the fact that the compliance requirements relevant to the order processing sub-process that are listed in Table1 covers all the basic types of compliance requirements (flow, temporal, data, resource). The answer to Q2 is ‘±’. Although the work performed in [GM05] proposed the transformation from FCL to RuleML [RuleML05] for runtime monitoring, the study assumes some extensions to RuleML, which have not been implemented. The answer to Q3 is ‘±’. Compliance checking during design time can be achieved on the basis of the *Idealness* notion introduced in [GMS06]. However, as mentioned in [GM05], RuleML does not support the use of modality and it is unable to deal with violations, although some improvements to deal with these issues are introduced but not implemented. Several rule engines have been proposed to execute or monitor the set of RuleML rules performance at runtime, such as jDREW [Spe02] and OO jDREW [BBH+05].

The critical concern related to FCL is the complexity of the language, which makes it difficult for compliance and business experts to utilise it. The graphical interface for the language is an open issue. In addition, FCL doesn’t provide an intelligible feedback mechanism that can help process experts to resolve non-compliances.

2.3.2. Evaluation of Temporal Logic Formalism

LTL as the representative language of this class of languages is a declarative language that supports non-determinism in terms of the guards (conditions). For example; in R6 in section 2.2.2, the atomic proposition *Action =Discount & paralist = {y, dis_limit}* plays the role of a guard indicating the condition of discount exceeding a predefined limit. If this guard is evaluated to true, the discount should be approved by the manager: *Action= Approve & paralist = {dis_limit, manager}*. LTL can be considered as generic because we could represent all the compliance requirements listed in Table 1. However, it was particularly efficient in specifying the flow (sequence) and temporal constraints. According to the work performed in [LMX07], the NuSMV2 model-checker used has the *counterexample tracing* facility that the business expert can use to resolve from the non-compliance, consequently meeting the ‘Intelligible feedback’ criteria.

Complexity of the LTL is also an issue for the compliance and business experts. However, as proposed in [LMX07], a graphical-based interface such as the Business Property Specification Language (BPSL) can be utilised as a remedy. The study in [LMX07] also proposes the transformation rules between BPSL and LTL. In addition, Dwyer’s property specification patterns [DAC98] can also be predefined in BPSL graphical interface. As also discussed in Section 2.2.2, the PROPOLS language introduced in [YMH+99] can also be utilised for this purpose.

LTL was not able to capture all the semantics of the compliance requirements listed in Table 1. Particularly, it was not able to capture the semantics of the CTD operator. It also does not provide any mechanism for consistency checks between rules and for meeting the non-monotonic concern. In LTL rules are monotonic. In addition, LTL does not provide a mechanism for the augmentation of business process models with compliance requirements (the *Symmetric Request* metric).

Regarding the three questions raised in Deliverable 6.1 [D6.1], For Q1, LTL was able to express all the compliance requirements of Table 1. The answer to Q2 is ‘±’; however, the work in [NS07] for runtime checking can be integrated by the work in [LMX07] for design time validation. But this requires a set of extensions that will be investigated in the coming deliverables:

- Extending the work performed in [LMX07] by pre-defining Dwyer's property patterns in the BPSL temporal graphical interface language.
- Extending the work performed in [NS07] with the notion of composite patterns introduced in [YMH+06] to be able to capture complex compliance requirements that might contain a conjunction, disjunction and any other logical connectors for runtime monitoring and enforcement.
- The mapping between LTL to the logical formalisms proposed in [NS07] for runtime monitoring and enforcement.

The answer to Q3 is 'Yes'. Design time validation can be achieved by the means of model-checkers, while runtime monitoring and enforcement can be handled by integrating the work performed in [NS07].

2.3.3. Evaluation of XSRL

XSRL was expressive enough to capture the semantics of the compliance requirements of the running example. It can also be considered as easier for experts to utilise as opposed to aforementioned languages. A graphical interface can also be provided.

XSRL does not provide support for checking the consistency between compliance rules. XSRL supports the non-monotonic requirement by means of 'prefer-to' operator. It is generic enough to represent different types of compliance requirements (sequence, temporal, data and resource perspective). However, it does not support the augmentation of business process models with compliance requirements and does not provide any facility for the 'Intelligible feedback' criteria.

For XSRL, the answer to Q1 is 'Yes'; since we could represent the entire compliance requirements relevant to the running example using the XSRL. We should note that XSRL lacks the support for design time checking and mainly proposes to address runtime monitoring. In that sense, Q2 is inapplicable for this case. The answer to Q3 is 'Yes' for the runtime monitoring part as XSRL provides an automated checking module based on constraint satisfaction principles, however no support is provided for the design time checking.

XSRL has a family of rich constructs ranging from sequence, modality and logical operators. It supports the equality comparisons (=, <, >), which is absent in FCL and LTL.

3. Compliance Request Language (CRL)

In accordance with the comparative analysis presented in section 2.3, the XSRL alternative can be eliminated, since it lacks support for design-time validation. Therefore, the decision should be determined between the deontic logic and temporal logic approaches.

The emphasis in deontic logic approaches is on compliance requirements emerging from business contracts. However, our main focus is on compliance requirements imposed by legislation and regulatory bodies, which might be different from that emerging from business contracts. The temporal logic approach has the following advantages over the deontic logic approach:

1. It is a mature field.
2. It has sophisticated verification tools tested and used since more than 20 years.
3. It is proved to be successful to verify large number of complex designs [Var01].

4. Property specification patterns can be defined, which can significantly facilitate the work of the business and compliance experts and reduce ‘complexity’.
5. It satisfies all essential criteria listed in Table 2. The criteria that lack support (such as the ‘consistency checking’ concern) are considered as future studies to be covered in the coming deliverables.

Two alternative languages for the temporal logic formalism to be considered as CRL are linear temporal logic (LTL) and computational tree logic (CTL). Vardi in [Var01] provides an informative comparison between LTL and CTL. Table 3 summarises the findings.

According to [Var01], CTL and LTL correspond to two distinct views of time, and consequently LTL and CTL are expressively incomparable. For example; the LTL formula FG_P (where F indicates eventually and G indicates always) is not expressive in CTL, whereas, the CTL formula $AFAG_P$ (where A is the universal path quantifier, F and G holds the same meanings) is not expressible in LTL. Although LTL and CTL are expressively incomparable from a theoretical point of view, from a practical point of view LTL is considered to be more expressive than CTL.

LTL is considered to be more intuitive than CTL. For instance; in LTL, the formula FX_P is equal to the formula XF_P, where both indicate that ‘p holds sometimes in the strict future’. In contrast, the two CTL formulas $AFAX_P$ and $AXAF_P$ are not logically equivalent; only the second formula ($AXAF_P$) indicates that ‘p holds sometimes in the strict future’. The un-intuitiveness of CTL significantly reduces the usability of CTL-based formal verification tools. From the verification point of view, CTL is considered to be more difficult that LTL due to the branching nature of CTL.

In computer system design, two types of systems are distinguished; ‘closed’ and ‘open’. A *closed* system’s behaviour is completely determined by the state of the system. While an *open* system interacts with its environment and its behaviour depends on this interaction. In the case of open systems, we have two types of non-determinism; *internal* non-determinism, which represents non-determinism stemming from the system, and *external* non-determinism, which represents the non-determinism stemming from the environment the system interacts with. In case of LTL, there is no distinction between internal and external non-determinism, while in CTL, both types of non-determinism should be dealt by different ways, and it is required to define a new model checking mechanism for open-systems.

		CTL	LTL
Expressiveness		Limited	Expressive
Intuitiveness		Unintuitive e.g.: AFAXp ≠ AXAFp	Less unintuitive e.g.= FXp = XFp
Difficulty		Difficult	Less Difficult
Internal and external non-determinism (Open systems)		Need to define new different model checking problem for open-systems	No distinction between Internal and external non-determinism
Model Checkers Complexity **n: size of the transition system* **m: size of temporal logic formula*	**Closed System**	O(nm)	$n.2^{O(m)}$
	Open System	EXPTIME-complete CTL*:2 EXPTIME-complete	PSPACE-complete
	Hierarchical, pushdown systems	PSPACE-complete Exponential in size	PSPACE-complete Polynomial in size
Compositional verification		Unsupported	Supported
Semi-formal verification (Model checking + simulation techniques)		Unsupported	Supported
Industrial usage		More widely used	Less widely used
Bisimulation		Supported	Unsupported
Counterexample facility		Not possible in some cases. E.g.: AFAXp	Usually possible (The ability to specify bounded length traces)
Uniformity		Non-uniform treatment of model-checking, abstraction and refinement	Uniform treatment of model-checking, abstraction and refinement (Language containment)

Table 3 Comparison between CTL and LTL

The main advantage of CTL over LTL is the computational complexity of its associated model-checkers. In CTL, the computational complexity is 'O(nm)', where n is the size of transition system and m is the size of the temporal logic formula. In LTL, on the other hand, the computational complexity is '$n.2^{O(m)}$'. However, Vardi [Var01] argues that LTL's complexity is not reflective, since it is based on worst cases. Hence, the complexity of LTL considered acceptable since the size of m is significantly smaller than the size of n. An abstraction of LTL properties leads to dramatic state space reduction. As proved by Vardi [Var01], this superiority disappears in the context of open systems.

Model checking is known to suffer from the *state-explosion* problem. This means that in a concurrent setting, the system is typically the parallel composition of many modules. As a result, the size of the state space of the system is the product of the size of the state spaces of the participating modules. This will lead to a huge state space, which makes model-checking algorithms impractical. Compositional verification is the technique that can be used to address this problem. It enables model checking to be applied only to the underlying modules, which have smaller state spaces. In contrast to CTL, compositional verification is supported by LTL.

For large systems, it is desirable to combine the model checking techniques with the simulation techniques due to the aforementioned state-explosion problem. The combination between formal model checking and informal simulation techniques is referred to as the *semi-formal* verification approach. Contrary to CTL, LTL provides support for semi-formal verification.

Bisimulation equivalence is the finest notion for the equivalence between two systems. Bisimulation equivalence is supported within the CTL context, in contrast to LTL. However, our focus is on behavioural properties rather than the structural ones. The counter example tracing facility is a significant functionality provided by the model-checkers that allows the verification engineers to focus on fragments of the system that cause the non-compliance. For the CTL, the counter example facility is not possible for some cases; whist for the LTL, the counter example facility is generally possible and the user can specify a bounded length of traces. As a final characteristic, LTL uniformly treats the system under consideration and the logical properties, where both of them can be mapped to the same transition system (such as Buchi automaton [Buc60]). The verification is performed by checking the emptiness of the intersection between the complement of the property system and the system under consideration. This approach is called the *language-containment* approach.

It is important to note that CTL* is the logic that combines the capabilities of both CTL and LTL. However, its computational complexity is higher than both of them.

Based on the comparison analysis summarised in Table 3, LTL is a better candidate for the CRL. However, it is important to note that with respect to the unique requirements of the project, it might be necessary to extend LTL in the future (in the coming deliverables for work package 2 [DOW]) or to integrate it to other types of logic to cover the basic types of compliance constraints.

As initially suggested in Deliverable 2.1 [D2.1], compliance requirements should be represented at various levels of abstraction to accommodate different stakeholders' needs. In particular, compliance requirements can be rendered as compliance goals, policies and rules. Figure 1 shows the relationship between the abstraction levels of compliance requirement representations corresponding to different stakeholders' perspectives.

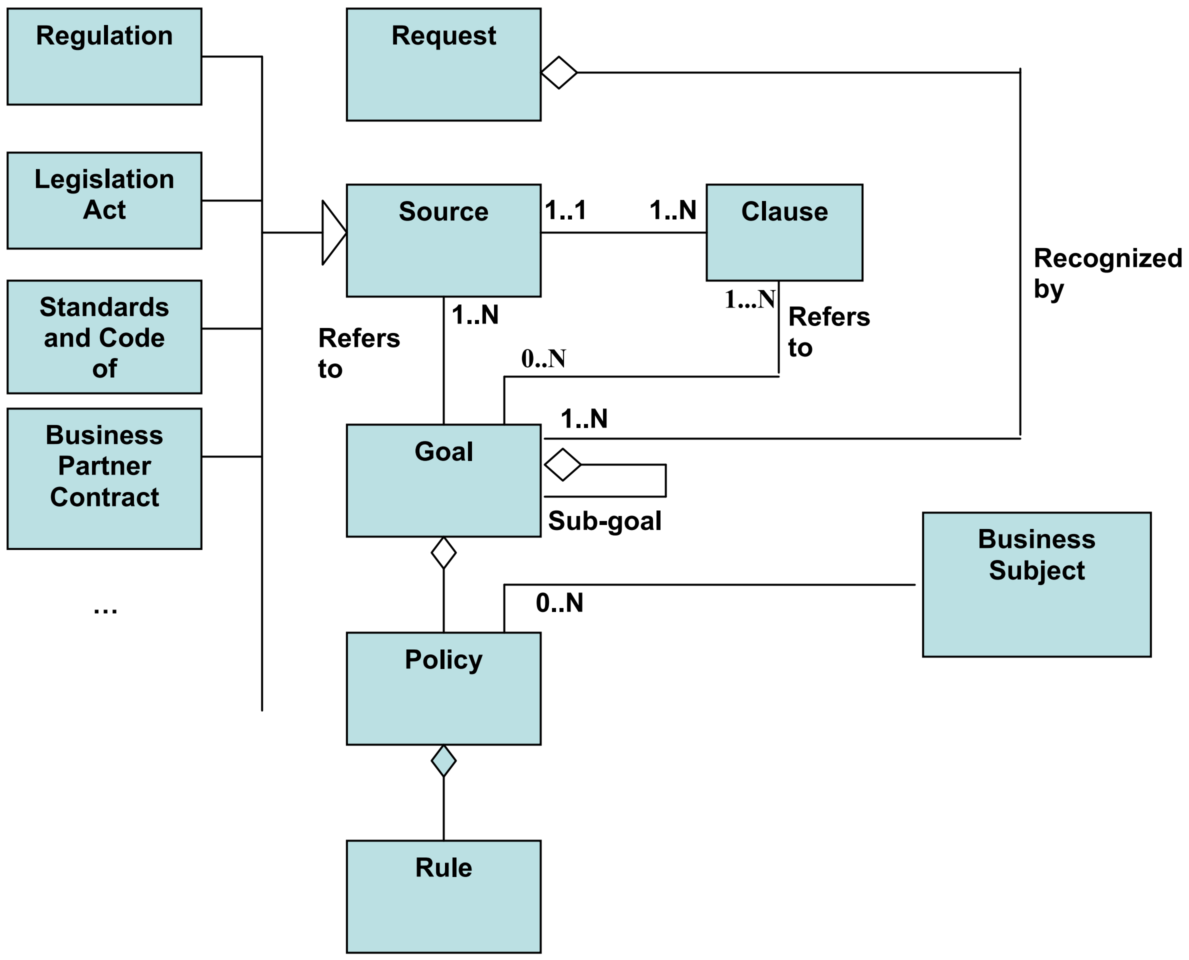

Figure 1 Compliance requirements representation

A *request* constitutes a compliance constraint or assertion, which may be defined at various levels of abstractions, in close collaboration with business analysts and compliance experts. E.g. Protect the loan origination process against fraud. A *request* allows the user to express and enforce his/her requirement with regard to particular business process. While the term *query* is used to retrieve information regarding instances of running business processes, some of which may be compliant to regulations and some may not. A request can be rendered as compliance goals, policies and rules.A *goal* is a high level and abstract description of the compliance requirement(s) a system has to satisfy in order to be deemed as compliant. E.g.: “Double check the customer’s credit worthiness by two different officers”, Implement a division of roles and responsibilities that reduces the possibility for a single individual to compromise a critical process”, “I want to comply with SOX Section 404.a and Basel II”. A goal refers to one or more source(s). A *source* is the origin of compliance requirements. A source consists of a set of clauses**.** A *clause* abstractly describes a compliance requirement on a business subject (business process or resource) that must be met.A source can be legislation and regulatory bodies (such as Sarbanes-Oxley [SOX2002], Basel II [BaselII2004]), standards and code of practice, or business partner contracts. A goal refers to one or more source(s). For example: the goal “I want to comply with SOX Section 404.a and Basel II” refers to SOX and Basel II. The goal also refers to one or more clauses in the source, e.g. the goal “I want to comply with SOX 404.a” refers to Section 404.a in SOX. A goal consists of a set of policies and a *policy* is a logical grouping of a set of coherent rules. At the lowest level of abstraction, a *rule* is a basic generalization that is accepted as true and can be used as a basis for reasoning or conduct. Rules usually take the form of If-Then statements. A rule has a unique identifier, consists of a set of conditions as its antecedents, and one or more conclusion(s). In this deliverable, our focus is on the Rule level, thus we followed a bottom-up approach.

Different languages and approaches that can be used for the representation and organization of higher levels of abstraction are subject to research and investigation in the coming deliverables. A *business subject* represents a business process, a business process fragment, a business process activity or a business service. Compliance and business specification are linked via the 'constrains' relationship. A policy can constrain one or more business subjects and the business subject can be constrained by zero or more policies. Since the primary emphasis in this deliverable is on the compliance specification perspective, the details of business process specifications are not represented and an example of specifying license compliance concerns is given in Deliverable 5.2 [D5.2].

In the following sub-section, a high-level framework for the compliance problem is presented to clarify how compliance requirements can be specified, represented, combined, checked on consistency, queried and matched against business process fragments from work package 4 [DOW]. Next, a meta-model for the initial CRL is presented. This CRL is subject to extensions in the coming deliverables.

3.1. Compliance Management Framework

Figure 2 shows a proposed compliance management framework from a compliance specification perspective. The purpose of the compliance framework is to present how compliance requirements can be represented, combined, checked for consistency, queried and matched against business process fragments.

The process involves various stakeholders including compliance and business experts. The compliance expert is typically an actor responsible for the specification and management of compliance requirements, stemming from external sources, such as legislations and internal sources, including company policies. The business expert is an analysts or a business process integrator who is responsible for defining and managing business processes in an organization, while taking into account compliance requirements. The current version of this framework focuses on design-time compliance aspects.

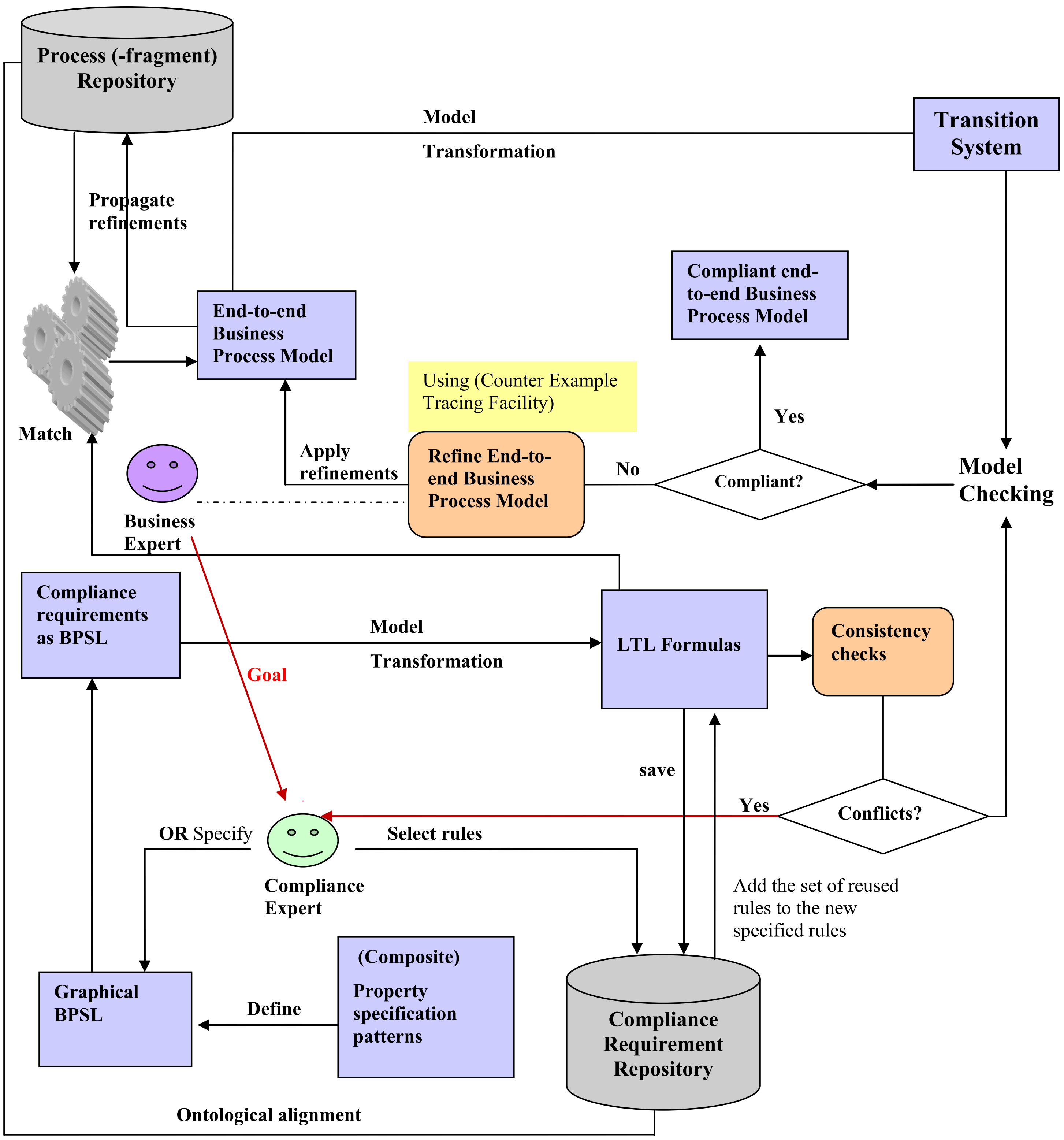
Process (-fragment) Repository
Model Transformation
Transition System
Propagate refinements
Compliant end-to-end Business Process Model
End-to-end Business Process Model
Using (Counter Example Tracing Facility)
Yes
Match
Apply refinements
Refine End-to-end Business Process Model
No
Compliant?
Model Checking
Business Expert
Compliance requirements as BPSL
Model Transformation
LTL Formulas
Consistency checks
Goal
save
Yes
Conflicts?
OR Specify
Select rules
Compliance Expert
Add the set of reused rules to the new specified rules
(Composite) Property specification patterns
Graphical BPSL
Define
Compliance Requirement Repository
Ontological alignment

Figure 2 A high-level compliance management framework

The process depicted in Figure 2 executes as follows: First, the business expert communicates abstract *compliance goals* to the compliance expert. The *compliance repository* stores and organises compliance requirements at various abstractions levels (in terms of goals, policies and rules) and allows the reusability of the compliance constraints. Various database technology and knowledge-based systems can be utilised in this context. Organizing compliance requirements in a repository is also important in handling the evolution of the compliance constraints. Appropriate mechanisms for versioning and propagating changes to the affected business processes should be taken into consideration in the coming deliverables of work package 2 and in other relevant work packages [DOW].

The compliance expert browses and queries the compliance repository to identify existing policies that realise the compliance goal. The control repository can support different querying capabilities. Query-By-Example (QBE) [SKS06] can also be utilised in this context. QBE queries are expressed “by example” instead of giving a procedure for obtaining the desired answer, in other words, the user gives an example of what is desired, and the system generalises this example to compute the specific answer for the query. For an example of a compliance goal “conform to SOX 402”, the compliance expert may search for associated policies in the repository, e.g., policies that realise segregation of duties.

If reusable policies are not found, the expert formalises the requirement (possibly with the help of a legal expert) and stores resulting policies in the repository. The framework proposes the *business property specification language (BPSL)* [LMX07] (with a graphical interface) to the compliance expert to facilitate this task. This language will be equipped with pre-existing compliance building blocks (named property specification patterns [YMH+06], [DAC98]) that can be composed into compliance rules (details of this part is illustrated in the next sub-section). The graphical BPSL specification will be automatically mapped to LTL to cater for logical inferences, where the automatic transformation methods proposed in [DAC98] and [LMX07] can be exploited.

The set of compliance rules can be a set of newly defined rules, a set of existing rules, or a combination between existing and new rules. The new specified rules are then stored in the compliance repository to enable their reusability.

The next task checks the *internal consistency* among the rules that have been selected or created by the compliance expert. This framework will consider *theorem proving* [RS01], [BKS+01] to address this challenge, which is considered as a future work. In case of conflicts, the compliance expert will be notified to reconcile the rules by selecting alternative rules from the repository or by communicating the conflict with the business experts to modify a particular compliance goal. The definition and consistency checking activities are performed in a highly iterative fashion -assuming active involvement from the compliance and business experts. These activities are finalised after the set of compliance rules reach to a consistent state. We assume that the compliance specifications (stored in the compliance repository) and business specifications (stored in the business process fragments repository) are defined based on the same domain ontology, represented by the ‘*Ontological alignment*’ relationship between the two repositories. Establishing an agreement on the terms, concepts and usage between the two requirements is a mandatory part for managing compliance to enable the matching between the set of compliance rules against the business process fragments. This matching can be a name-based matching, which selects the set of business process fragments from the business process fragments repository whose activities are mentioned in the set of compliance rules.

Next, selected business process fragments are composed into *end-to-end business processes*. Building the compositions can be performed manually or automatically. (The framework currently assumes a manual composition by the business expert. Although building such compositions automatically is an interesting research subject, it is out of the scope of work package 2 [DOW]). The composed fragment constellation is verified. To enable the usage of automatic model checkers, end-to-end business processes should be mapped to some variant of transition systems, such as FSM and Buchi automaton [Buc60]. This transformation can be performed automatically, for example, exploiting the work performed in [LMX07] for NuSMV2 or in [AM07]. In case of conflicts, the compliance expert is expected to make changes to either the business process fragments by the help of the counter-example tracing facility or the compliance rules. Again, these activities are performed in a highly iterative manner until a stable and consistent end-to-end process has been defined. In the next sub-section we present the meta-model of the proposed CRL, of which LTL forms the core logic.

3.2. Meta-Model for the CRL

The proposed meta-model depicted in Figure 3 is an integration of the work performed in [DAC98], [YMH+06] and based on LTL formulas. The notion of property specification patterns was introduced in [DAC98] by Dwyer. Temporal logic formulas in general are difficult to write and understand for users lacking solid mathematical backgrounds [YMH+06]. Dwyer's property patterns are high-level abstractions of frequently used temporal logic formulas. The property patterns help users to read and write formal specifications and consequently help bridging the gap between model checkers and end-users. The mapping from property specification patterns to various temporal logic formulas, namely LTL, CTL and QRE was also proposed in [DAC98]. In [DAC99] a survey was carried out over 500 examples of property specifications, which revealed that majority of the examples are instances of the proposed pattern system. The authors also introduced extensions to accommodate new patterns and variations of existing patterns encountered in the survey.

The work in [YMH+06] extends Dwyer's property specification pattern by introducing the logical composition of patterns. This mechanism enables the definition of complex requirements in terms of property patterns. The part of the Figure 3 depicting the pattern system is based on the extended property specification patterns system proposed in [YMH+06]. A property specification pattern consists of a pattern and a scope. The *pattern* specifies what must occur and the *scope* specifies when the pattern should occur. Patterns are classified into occurrence patterns and order patterns. Occurrence patterns are (P and Q being a given state/event):

- *Absence*: Indicates that P does not occur within a scope.
- *Universality*: Indicates that P occurs throughout the scope.
- *Existence*: Indicates that P must occur within a scope
- *Bounded existence*: Indicates that P must occur at least/exactly/at most K times within a scope.

Order patterns are (P and Q being a given state/event):

- *Precedence*: P must always be preceded by Q within a scope.
- *Response*: P must always be followed by Q within a scope.
- *Chain precedence*: A sequence of states/events P1, … Pn must always be preceded by a sequence of states/events Q1, … Qm.
- *Chain response*: A sequence of states/events P1, … Pn must always be followed by a sequence of states/events Q1, … Qm.

The scope defines a starting and an ending state/event for a pattern. A scope can be (P and Q being a given state/event):

- *Globally*: The pattern must hold during the complete system execution.
- *Before*: The pattern must hold up to the occurrence of a given P.
- *After*: The pattern must hold after the occurrence of a given P.
- *Between*: The pattern must hold from the occurrence of a given P to the occurrence of a given Q.
- *Until*: Has the same meaning as 'Between' but the pattern holds even if Q never occurs.

The semantics of the pattern properties can be specified using FSA [DAC98]. An 'Expression' (in Figure 3: Pattern System) is a general class for patterns and operations, and all patterns and scopes are the sub-classes of 'Unary' or 'Binary'. Therefore, they share the properties defined in Unary or Binary; either has an 'operand' property pointing to expression or has a firstOperand property and secondOperand property. Further restrictions were also considered in [YMH+06]. The notion of 'compositePattern' was introduced using Boolean logic operators including Not, And, Or, Xor and Imply, which enables the definition of complex requirements in terms of property patterns. The correctness of this composition was also proved in [YMH+06]. Figure 4 depicts the composite pattern structure, where the new 'PLeadsTo' pattern was introduced. 'PLeadsTo' corresponds to a frequently used composite pattern indicating "P Precedes Q And P LeadsTo Q". (Note that in [YMH+06], "P LeadsTo Q" is used, which is logically equivalent to "Q respondTo P" in Dwyer's pattern system.)

As shown in Figure 3, an automatic mapping can be performed from a property specification pattern expression to LTL rules as proposed in [DAC98], and subsequently model checkers can be utilised. As we proposed in Figure 2, the property specification pattern system can be defined in a graphical tool such as the business property specification language (BPSL) used in [LMX07]. The study in [LMX07] also proposes a mapping from BPSL to LTL formulas to facilitate the definition.

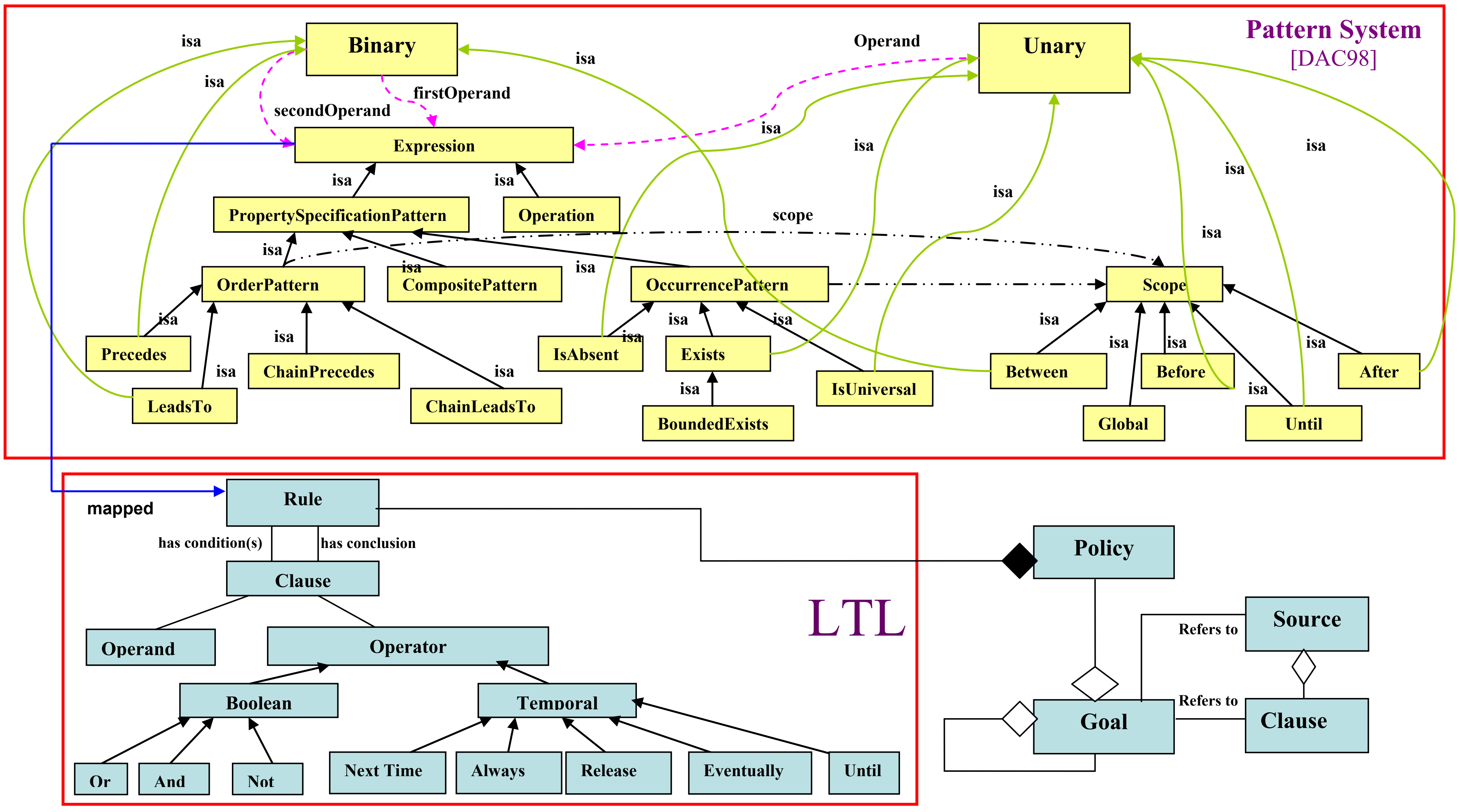

Figure 3 Meta-Model for the CRL

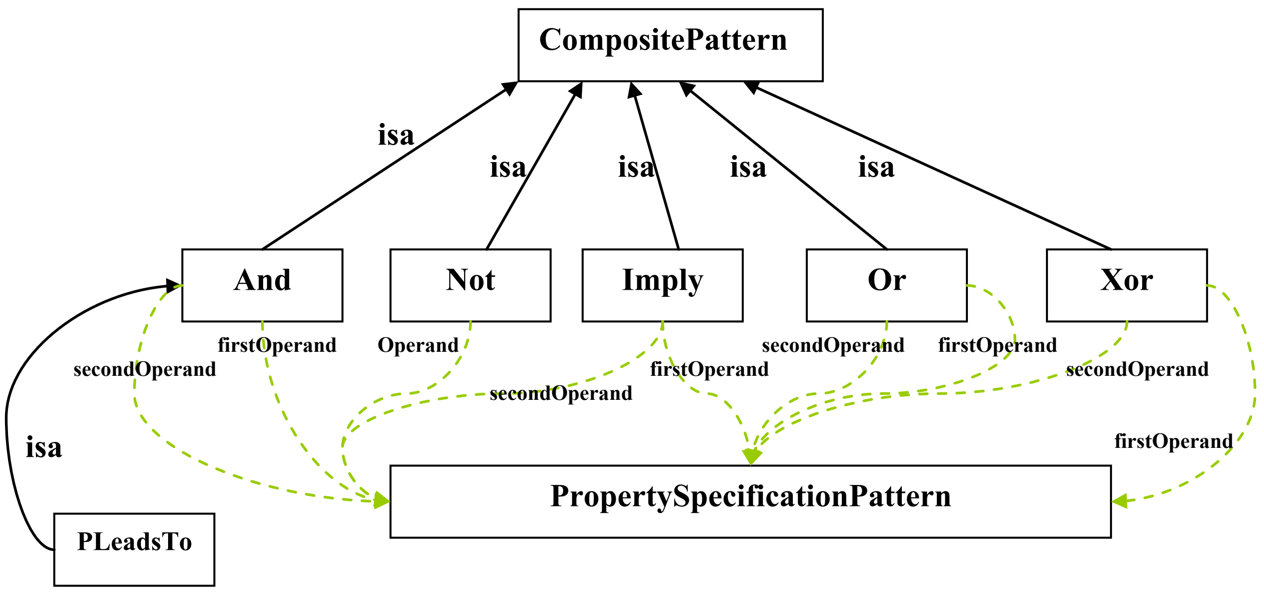

Figure 4 Composite Pattern Structure (Based on [YMH+06])

4. Conclusions and Open Issues

In this deliverable an initial CRL is proposed for the formal specification of compliance requirements based on the extended Dwyer's property specification pattern system and LTL formulas. A meta-model for the CRL is also proposed as an initial model, which is subject to extensions and modifications in the future to accommodate further requirements relevant to compliance management. A high-level compliance management framework is proposed from a compliance specification perspective. There are various open issues that are subject for future research and investigation. These issues can be summarised as follows:

1. We advocated in this deliverable and in Deliverable 2.1 [D2.1] the need to represent compliance requirements at various levels of abstraction to accommodate with different stakeholders' requirements. The main focus of this deliverable was on the 'Rules' abstraction level. Higher abstraction levels, namely: requests, goals and policies, should be studied and investigated for their representation and organization.
2. Checking consistency among compliance rules is an important concern that should be considered in the coming deliverables. Contradictions and conflicts may arise between compliance requirements particularly if they originate from different sources. Theorem proving techniques and the work performed in [JH05] are possible candidates for a resolution.
3. The meta-model proposed does not support the specification of non-monotonic rules. A solution to this problem should be considered in the future.
4. A survey, similar to the one carried out in [DAC99], over the examples of *property specifications*, should be conducted from a compliance point of view to uncover required extensions to accommodate relevant requirements.
5. This work focused on design-time validation. The work in [NS07] should be assessed for runtime monitoring to introduce a mapping between the proposed CRL and the

formalisms used in [NS07]. The work in [NS07] is also based on Dwyer's properly specification patterns, but it focuses on runtime monitoring.

5. Reference documents

5.1. Internal documents

[DOW]	Description of Work
[D2.1]	State-of-the-art in the Field of Compliance Languages
[D4.1]	State-of-the-art for Improving Reusability of Processes and Services Compositions
[D5.2]	Initial goal-oriented data model
[D6.1]	Use Case, Metrics and Case Study

5.2. External documents

[AM07] F.Abouzaid,, and J.Mullins, "A Calculus for Generation, Verification, and Refinement of BPEL Specifications," *Proc. 3rd Int'l Workshop on Automated Specification and Verification of Web Sites* (WWV'07), Italy, 2007, pp. 43-65.

[APY+02] M.Aiello, M.Papazoglou, J.Yang, M.Carman, M.Pistore, L.Serafini, P.Traverso, "A Request Language for Web Services Based on Planning and Constraint Satisfaction," *Proc. VLDB Workshop on Technologies for E-Services*, 2002, pp. 9-38.

[BaselII2004] Bank for International Settlements, "*Basel II: International Convergence of Capital Measurement and Capital Standards: a Revised Framework*", 2004.

[BBH+05] M.Ball, H.Boley, D.Hirtle, J.Mei, B.Spencer, "Implementing RuleML Using Schemas, Translators, and Bidirectional Interpreters," *Proc. the W3C Workshop on Rule Languages for Interoperability*, Position Paper, 2005.

[Buc60] K.Buchi, "On a Decision Method in Restricted Second Order Arithmetic," *Proc. Int'l the Int'l Congress on Logic, Method, Philosphy of Science*, Stanford, 1960, pp. 1-11.

[DAC98] M.Dwyer, G.Avrunin, J.Corbett, "Property Specification Patterns for Finite-State Verification," *Proc. 2nd Int'l workshop on Formal Methods on Software Practice*, USA, 1998, pp. 7-15.

[DAC99] M.Dwyer, G.Avrunin, J.Corbett, "Patterns in Property Specifications for Finite State Verification," *Proc. the Int'l Conf. on Software Engineering*, USA, 1999, pp. 411-420.

[FINRA08] The Financial Industry Regulatory Authority, "*FINRA Manual*", Dec.2008, http://www.finra.org/.

[Gov05] G.Governatori, "Representing Business Contracts in RuleML," *International Journal of Cooperative Information Systems*, 2005, pp.181-216.

[GLM+05] C.Giblin, A.Liu, S.Muller, B.Pfitzmann, and X.Zhou, "Regulations Expressed As Logical Models," *Proc.18th Int'l Conf. of legal knowledge and information systems*, Belgium, 2005.

[GM05] G.Governatori, and Z.Milosevic, "Dealing with Contract Violations: Formalism and Domain-Specific Language," *Proc. 9th Int'l IEEE Conf.* (EDOC), The Netherlands, 2005.

[GMS06] G.Governatori, Z.Milosevic, and S.Sadiq, "Compliance Checking Between Business Processes and Business Contracts," *Proc. 10th Int'l Enterprise Distributed Object Computing Conf.* (EDOC 2006), 2006.

[GRS05] G.Governatori, A.Rotolo, and G.Sartor, "Temporalised Normative Positions in Defeasible Logic," *Proc. 10th Int'l Conf. on Artificial Intelligence and Law* (ICAIL), 2005, Italy.

[GV06] S.Goedertier, and J.Vanthienen, "Designing Compliant Business Processes with Obligations and Permissions," *Proc. the Business Process Management Workshops* (BPM), Austria, 2006.

[IFRS01] International Accounting Standards Board, "*International Financial Reporting Standards,*" 2001, http://www.ifrs.com/.

[JH05] Y.Jin, J.Han, "Consistency and Interoperability Checking for Component Interaction Rules," *Proc. 12th Int'l Asia-Pacific Software Engineering Conf.*, Taiwan, 2005.

[LAP06] A. Lazovik, M.Aiello, and M.Papazoglou, "Planning and Monitoring the Execution of Web Services Requests," *International Journal on Digital Libraries*, vol. 6, no. 3, 2006, DOI: 10.1007/s00799-006-0002-5.

[LGR+08] L.Thao Ly, K.Goser, S.Rinderle-Ma, P.Dadam, "Compliance of Semantic Constraints- A Requirements Analysis for Process Management Systems", In Proc. 1st Int'l Workshop on Governance, Risk and Compliance - Applications in Information Systems (GRCIS'08), France, 2008.

[LMX07] Y.Liu, S.Muller, and K.Xu, "A Static Compliance-Checking Framework for Business Process Models," *IBM Systems Journal*, vol. 46, no. 2, 2007, pp. 335 – 361, DOI: 10.1147/sj.462.0335.

[LSG07] R.Lu, S.Sadiq, G.Governatori, "Compliance Aware Business Process Design," *Proc. 5th Int'l Conf. on Business Process Management* (BPM 2007), Brisbane, 2007.

[NS07] K.Namiri, N.Stojanovic, "A Model-driven Approach for Internal Controls Compliance in Business Processes," *Proc. the Semantic Business Process Management Workshop* (SBPM), 2007.

[RS01] M.Roggenbach, L.Schröder, "Towards Trustworthy Specification I: Consistency Checks," *Lecture Notes in Computer Science*, vol. 2267, Springer Berlin / Heidelberg, 2001, pp. 305 – 327.

[RuleML05] "RuleML:The Rule Markup Initiative", February 2005, http//www.ruleml.org

[SGN07] S.Sadiq, G.Governatori, K.Naimiri, "Modeling Control Objectives for Business Process Compliance," *Proc. 5th Int'l Conf. on Business Process Management*, Australia, 2007.

[SOX2002] "Sarbanes-Oxley Act", July 2002, http://www.sarbanes-oxley.com/section.php?level=1&pub_id=Sarbanes-Oxley.

[SKS06] A.Silberschatz, H.Korth, S.Sudarshan, *Database System Concepts*, McGrawHill, 2006, pp. 171-180.

[Spe02] B.Spencer, "The Design of j-DREW: A Deductive Reasoning Engine for the Web," *Proc. 1st Int'l CologNET Workshop on Component-Based Software Development and Implementation Technology for Computational Logic Systems* (CBD ITCLS), Spain, 2002.

[Var01] M. Vardi, "Branching vs. Linear Time: Final Showdown," *7th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, Italy, 2001, LNCS 2031, Springer Berlin / Heidelberg, 2001, pp. 1-22.

[YMH+06] J.Yu, T.Manh, J.Han, and Y.Jin, "Pattern-Based Property Specification and Verification for Service Composition," *Web Information Systems* (WISE), 2006.