Softw Syst Model (2016) 15:119–146
DOI 10.1007/s10270-014-0395-3

REGULAR PAPER

Formalizing and appling compliance patterns for business process compliance

Amal Elgammal · Oktay Turetken · Willem-Jan van den Heuvel · Mike Papazoglou

Received: 9 May 2013 / Revised: 9 December 2013 / Accepted: 5 January 2014 / Published online: 8 February 2014

Abstract Today's enterprises demand a high degree of compliance of business processes to meet diverse regulations and legislations. Several industrial studies have shown that compliance management is a daunting task, and organizations are still struggling and spending billions of dollars annually to ensure and prove their compliance. In this paper, we introduce a comprehensive compliance management framework with a main focus on design-time compliance management as a first step towards a preventive lifetime compliance support. The framework enables the automation of compliance-related activities that are amenable to automation, and therefore can significantly reduce the expenditures spent on compliance. It can help experts to carry out their work more efficiently, cut the time spent on tedious manual activities, and reduce potential human errors. An evident candidate compliance activity for automation is the compliance checking, which can be achieved by utilizing formal reasoning and verification techniques. However, formal languages are well known of their complexity as only versed users in mathematical theories and formal logics are able to use and understand them. However, this is generally not the case with business and compliance practitioners. Therefore, in the heart of the compliance management framework, we introduce the Compliance Request Language (CRL), which is formally grounded on temporal logic and enables the abstract pattern-based specification of compliance requirements. CRL constitutes a series of compliance patterns that spans three structural facets of business processes; control flow, employed resources and temporal perspectives. Furthermore, CRL supports the specification of compensations and non-monotonic requirements, which permit the relaxation of some compliance requirements to handle exceptional situations. An integrated tool suite has been developed as an instantiation artefact, and the validation of the approach is undertaken in several directions, which includes internal validity, controlled experiments, and functional testing.

Keywords Business process compliance · Compliance patterns · Formal specification · Regulatory compliance · Compliance management tool support · Design-time compliance management

Communicated by Prof. Ulrich Frank.

A. Elgammal (✉)
Governance, Risk Management and Compliance Technology Centre (GRCTC), University College Cork, Cork, Ireland
e-mail: aelgammal@ucc.ie

O. Turetken
School of Industrial Engineering, Eindhoven University of Technology, 5600 MB Eindhoven, Netherlands
e-mail: o.turetken@tue.nl

W.-J. van den Heuvel · M. Papazoglou
European Research Institute in Service Science (ERISS), Tilburg University, 5000 LE Tilburg, Netherlands
e-mail: w.j.a.m.vdnheuvel@uvt.nl

M. Papazoglou
e-mail: m.p.papazoglou@uvt.nl

1 Introduction

In the light of the recent high-profile business scandals and failures, such as Enron and Countrywide Financial, today's business entities are faced by an ever-growing number of laws and regulations, such as Sarbanes-Oxley [1] and Basel III [2]. These require organizations to audit their business processes and ensure that they meet compliance requirements set forth in laws and regulations. Without explicit business process (BP) definitions and effective internal control structures, organizations may face litigation risks and even

criminal penalties. In 2012, banks had to pay exponentially increasing fines as a result of their failure to show their full compliance with anti-money laundering directives [3]. Nine international institutions were sanctioned (e.g. HSBC, Standard Chartered, ING, ABN Amro), where HSBC had the highest rank of 1,920 million dollars of a fine. Executives and analysts of diverse industry sectors identified regulation and compliance as the top business risk [4,5].

We define *compliance* as the process of ascertaining the adherence of business processes and applications to relevant compliance requirements, which may emerge from laws, legislation, regulations, standards and code of practices (such as ISO 9001), internal policies, and business partner contracts (such as service level agreements—SLAs). From a structural perspective, compliance requirements may fall into four classes that pertain to the basic structure of business processes, which are [6] as follows: (i) workflow constraints (control-flow requirements), (ii) information usage (data validation and requirements), (iii) employed resources (task allocation and access rights), and (iv) temporal constraints.

With the ultimate objective of having absolute compliance assurances, many organizations implement compliance solutions on an ad hoc and per case basis. These solutions are generally handcrafted for a particular compliance problem, which creates difficulties for reuse and evolution. These solutions usually lack the flexibility needed to rapidly adapt to ever-changing business imperatives, as they usually involve hard-coded implementations across multiple systems. This makes it difficult to verify and ensure continuous guaranteed compliance. Having recognized these problems and their implications for organizations and firms, major enterprise software vendors have developed commercial products that provide a bundled set of compliance solutions (e.g. Oracle GRC Accelerators, SAP BusinessObjects, and GRC IBM OpenPages). However, these commercial products usually suffer from being highly proprietary (vendor lock-in) and technology-specific as they are usually tightly interweaved with pre-existing enterprise systems.

Efficiently addressing these problems represents an emergent *business need* that fundamentally requires a framework for managing business process compliance. This framework should be sustainable throughout the complete business process life cycle. A *preventive* focus is essentially required such that compliance is considered from the early stages of the business process design, thus enforcing *compliance by design* [7], which must further be taken up by compliance monitoring and adaptation of the running process instances. Accordingly, we consider design-time and runtime compliance checking and monitoring to be complementary and indispensable to ensure preventive compliance. This responds to today's business environment that strives for a pro-active compliance approach; where a violation is avoided and in case it occurs, recovery action(s) should (semi-)automatically take place to mitigate/minimize the impacts of these violations. In that respect, the main focus of this paper is on *design-time aspects* of BP compliance as a first key step towards a preventive compliance support

It is generally acceptable that compliance requirements should be based on a formal foundation of a logical language to enable automatic reasoning and analysis that assist in verifying and ensuring business process compliance in a sound, complete, and predictable manner. However, it is a well-known phenomenon that the use of formal languages creates difficulties for end-users, particularly in terms of usability and comprehensibility. This problem represents one of the main obstacles for the utilization of sophisticated verification and analysis tools associated with these languages. To surmount such problems, in the heart of the framework, we propose the Compliance Request Language (CRL) for the abstract specification of compliance requirements that is grounded on temporal logic, more specifically linear temporal logic (LTL). CRL incorporates and integrates a series of *compliance patterns*, which supports shielding the complexity of logical formalizms from business and compliance experts and facilitates their abstract specification. By applying the proposed mapping scheme, formal LTL rules, capturing applicable compliance requirements, can be automatically generated and directly forwarded to associated verification tools (i.e. model checkers [8]) for automated compliance assurance.

The contributions presented in this paper can be summarized as follows:

- We propose a *design-time* compliance management framework. The heart of the framework is the Compliance Request Language (CRL), which is a visual pattern-based language that is grounded on LTL and encompasses a series of *compliance patterns* in its core. *Compliance patterns* are high-level abstractions of frequently used compliance requirements, which help non-technical users to abstractly represent desired properties and constraints. They span three out of four structural classes of business process aspects; control flow, employed resources, and temporal (data/information aspect is left as a future work). As will be shown later in this paper, this enables to address a wide range of compliance requirements in different application domains.
- Furthermore, CRL provides two efficient mechanisms to allow the relaxation of some less-stringent compliance requirements, i.e. non-monotonicity and compensations.
- To investigate the applicability and expressiveness of CRL, we have developed an integrated software tool suite[1] and employed it in two case studies that deal with

[1] *BPCM* Business Process Compliance Management Tool Suite: http://eriss.uvt.nl/compas/.

business processes of companies in two different industry sectors.

The reminder of this paper is organized as follows: Related work is discussed in Sect. 2. Section 3 summarizes the overall approach for design-time compliance management, pinpointing the main focus of this paper. Section 4 introduces the case studies conducted and gives a simplified scenario used as a running example throughout this paper. Section 5 discusses the compliance request language in detail. Section 6 presents the software toolset as a prototypical implementation of CRL and proof- of-concept. The evaluation and the findings of the case studies are discussed in Sect. 7. Conclusions and a set of lessons that are gained from our research and development experience are portrayed in Sect. 8. Finally, ongoing and future work is highlighted in Sect. 9.

2 Related work

In the area of design-time business process compliance management, we discuss the related work in two broad categories; *temporal* logic approaches and *Deontic* logic approaches, corresponding to the formal model of compliance requirements. Key work examples from each class are discussed below and compared to the approach proposed in this paper. In addition to this categorization, we also discuss key graph pattern matching approaches that can also be applied in the context of BP compliance.

An *abstract* compliance management framework is provided in [9] that incorporates the fundamental requirements for a comprehensive lifetime support of compliance constraints. This framework can serve as a reference model for the open research challenges in this area. In this article, we address the research challenges that are related to design-time compliance management and propose solutions to address these issues. The research problems that we consider will be discussed in detail in Sect. 3, pinpointing the main focus of this paper.

2.1 Approaches based on temporal logic

Temporal logic has been successfully utilized in the literature to formalize and reason about the correctness of software and hardware designs and their adherence to desired properties and constraints in diverse application domains.

Proposals based on temporal logic follow either a bottom-up or a top-down verification approach. The core idea underpinning *bottom-up* approaches is on one side, to use a logical language (e.g. linear temporal logic—LTL, computational tree logic—CTL) to formally represent compliance requirements; and is on the other side, to transform low-level business process specifications (e.g. modelled using BPEL—Business Process Execution Language) to a corresponding formal representation (e.g. a finite state automata, Petri net, or a process algebraic representation). Next, model checking and other verification techniques are utilized to verify the compliance between the two specifications.

For example, in [10], an extension to computational tree logic (CTL) is proposed to capture data-dependent constraints, i.e. CTL-FO+. Next, CTL-FO+ formulas are mapped into pure CTL formulas and NuSMV model checker is utilized to ensure the conformance against BPEL specifications. In [11], real-time temporal object logic is utilized for the formal specification of compliance requirements based on a pre-defined domain ontology. Real-time temporal logic is a powerful logic; however, no verification approach is provided for design-time compliance checking. In [12], UML activity diagrams is considered, and then, PLTL (past LTL) is exploited to capture desired properties and constraints. Similarly, [13] utilizes first-order logic (FOL) to formally capture both workflow systems and desired constraints for the detection of any constraint anomalies.

In *top-down* temporal logic approaches, business process models are first represented using an abstract high-level language. Compliance requirements and desired properties are formalized using a temporal logic language. Next, model checkers and other verification techniques are also utilized for compliance checking. If the abstract BP model is compliant with the set of compliance rules, a corresponding BP model may be (semi-)automatically synthesized, e.g. as a BPEL or BPMN (Business Process Modelling and Notation) model. For example, [14] employs π-logic to formally represent compliance requirements. BP models are abstractly modelled using BP-calculus, which is a formal business process modelling language based on π-calculus. If business and compliance specifications are compliant, an equivalent BPEL program is automatically generated from the abstract BP-calculus model.

Similarly, authors in [15] follow a top-down approach such that LTL is exploited as the formal foundation of compliance requirements. From a set of LTL rules (capturing applicable compliance requirements) and by applying process-mining techniques, process templates are synthesized semi-automatically. Similarly, a business process synthesis approach is proposed in [16] based on temporal business rules.

The framework proposed in this article falls in this category and follows a bottom-up approach. We differentiate our work by the fact that the above approaches assume experienced users who are familiar with formal languages and mathematical theories, which is hardly the case in practice. To overcome the usability concern of formal languages, which represents the main obstacle of the utilization and application of sophisticated formal analysis and verification tools,

we introduce a series of high-level (visual) compliance patterns as an intermediate abstract representation to their corresponding formal rules. In addition, none of these approaches addresses three structural classes of compliance requirements in a single unified approach, e.g. in [11] workflow and real-time constraints are supported, in [10] workflow and data requirements are addressed. Moreover, compensations and non-monotonic requirements, which enable the handling of exceptional situations, are also not addressed in these works. Additionally, we consider Business Process Execution Language (BPEL), which is the standard language for the specification of executable and abstract business processes.

2.1.1 Graphical/pattern-based approaches

To address the usability concern of temporal logic, several research efforts propose pattern-based and/or graphical languages to model desired properties and constraints (e.g. [17,18] and [19]). Authors in [18] propose the BPMN-Q language, which extends BPMN to graphically represent compliance requirements analogously to the way business processes are modelled in BPMN. BPMN-Q is then mapped into past LTL (PLTL) and a model checking approach is utilized. Similarly, Extended Process Pattern Specification Language (EPPSL) is proposed in [20] focusing on quality constraints.

The study in [21] utilizes Dwyer's property specification patterns [22] for the verification of service compositions and introduces the logical compositions of these patterns via Boolean operators. The CHARMY framework is proposed in [23] for modelling and verifying designs in UML. CHARMY uses Property Sequence Chart (PSC) as a visual scenario-based language for representing required properties and constraints, which is then mapped into LTL formulas.

In [24], a compliance checking approach is proposed that utilizes a set of Petri net patterns. The compliance checking is performed by means of aligning these patterns with event logs of completed business process executions. This approach focuses mainly on offline business process monitoring of completed business process instances. The same path is adopted in [25] focusing on cloud-based business processes for the purpose of compliance certification, where a number of Petri net patterns are introduced. In [26], Petri net is used focusing on information flow analysis of business process models.

The approach we propose in this paper is closely related to the efforts discussed above. However, our work introduces a wider set of compliance patterns coupled with a graphical notation to enhance its usability. Besides, only workflow and data requirements classes are considered in the above approaches. We also consider design-time compliance checking as first key step towards preventive compliance support. Furthermore, non-monotonicity is not addressed in any of these works. Compensations is only addressed in [17].

DECLARE [27] and ConDec [28] are graphical pattern constraint languages for the declarative specification of workflow systems to improve their flexibility and to enable the management of dynamic processes. DECLARE and ConDec are translated into LTL and are built on the concept of workflow patterns.

While we share the same interest of using a high-level pattern-based language for constraint specification, the approach we propose in this paper considers the BPEL standard. Furthermore, the language we propose (CRL) incorporates a wider range of compliance patterns that supports the employed resources and temporal facets of business process specifications.

The specification and verification of temporal constraints is an important aspect in various application domains, e.g. real-time embedded systems. Studies in [29,30] and [31] extend Dwyer's property specification patterns [22] with real-time related patterns. For example, in [29], five real-time patterns are proposed based on an analysis performed on real-time requirements emerging from real-time embedded systems. The timed pattern class as one of the integral parts of the language (CRL) introduced in this paper supports a wider range of timed constraints. Furthermore, these studies only focus on the temporal aspect, and no support is provided for non-monotonic requirements and compensations.

2.1.2 Resource allocation approaches

The modelling and verification of task allocation and authorization constraints have also gained significant interest, particularly in the information systems security field. Task allocation and authorization constraints represent one of the four structural aspects of compliance requirements. Some studies in this area merely focus on the modelling and visualization of authorization constraints inside business process models, without offering a means for their verification. Works in [32] and [33] fall into this category. Wolter et al. [32] propose extensions to BPMN notations to enable the modelling and visualization of task-based authorization constraints.

Our approach is distinguished by having a rigorous verification method to reason about compliance requirement violations. We also agree with the argument in [7] that business and compliance specifications are to be handled separately if we aim to automatically verify their compliance. This is mainly because business and compliance specifications have different objectives (business perspective vs. ownership and governance perspectives), nature (procedural vs. declarative), and lifecycles and may conflict with each other resulting in a highly complicated BP models. Therefore, it should be possible for any of these specifications to be treated independently. However, it is also necessary for these two

specifications to be aligned, and their interconnections are established and maintained carefully. This can be possible by having a close collaboration between compliance experts and business process designers, which could be effectively supported by tools that enable sharing and aligning concepts and constructs that are common to both specifications.

In this context, our current work[2] focuses on the development of a *compliance management knowledge base* (CMKB) that incorporates and interrelates a set of ontologies capturing various perspectives of the compliance and business spheres. This will further facilitate the communication between different stakeholders with diverse backgrounds, as it will also be possible to annotate business process models with related compliance requirements and visualize them in a unified platform. For the purpose of the main objective of this article, i.e. automating the verification of business process models against applicable compliance requirement in a user-friendly manner, we assume that compliance and business specifications exist independently yet be aligned carefully.

2.2 Approaches based on deontic logic

Deontic logic is also common in specifying compliance constraints, especially in the context of business partner contracts, e.g. SLAs. Key works include [6,7] and [34]. The study in [34] provides the foundations of the FCL (formal contract language) focusing on business partner contracts. Compliance between business and compliance specifications can be verified based on the *idealness* notion [34]. In [7], FCL is used to express different types of requirements emerging from laws and regulations. FCL could express compliance requirements in the four compliance structural classes. FCL also supports the specification and verification of non-monotonic requirements.

In [35], an extension is proposed to FCL to incorporate real-time compliance dimension, i.e. temporalized violation logic. LKIF rule language [36] is extended with defeasible logic (FCL) in [37] for the formal specification of compliance requirements. The work in [38] proposes to extend FCL with the notion of goals (i.e. *goal compliance*), such that BP models could satisfy at the same time the goals of the organization, and the compliance requirements governing the business. In particular, they extend and combine FCL for modelling contracts and regulatory compliance, and the defeasible BIO (Belief–Intention–Obligation) logic for modelling agents [39]. In [38], first, an *abstract* BP model is specified and checked for compliance and goals satisfaction, which is followed by the automated generation of corresponding concrete compliant BP model.

[2] *Ongoing work in Governance, Risk and Compliance Technology Centre (GRCTC), Ireland,* http://www.grctc.com/, *the first author is affiliated to.*

The approaches referred above implicitly assume users who are experienced in formal theories and definitions. Furthermore, associated verification techniques are not as mature and heavily experimented as the sophisticated verification and analysis techniques associated with temporal logic

2.3 Graph pattern matching approaches

Designing and re-designing business processes from scratch is a highly complex, time-consuming, and error-prone task [40]. An important challenge is to enable the querying of business process artefacts for reusability purposes. This can be basically achieved through the exploitation of graph matching techniques. Dominant studies in this direction are [40–43], and [44]. Although the motivation underlying many of these approaches is to propose solutions for problems regarding process similarity and process substitution, it has been shown in [41] and [44] that these solutions could also be employed for compliance checking.

BP-QL [41] is a novel query language for querying business processes. It is an abstraction of the BPEL standard that allows users to query business processes visually in a way analogous to how business processes are specified. Compliance requirements are represented as queries in BP-QL, which describe the pattern of activities/dataflow of interest, and then, a graph matching approach is utilized. Similarly, the study in [42] proposes an approach that applies graph-based rules for identifying problems in business process models.

A pattern specification and matching approach is proposed in [43], which is based on the set theory and operations. The approach is generic in the sense that it doesn't assume a specific modelling language, such that any conceptual model is treated as a graph. The approach only focuses on the specification and matching of structural patterns. The work presented in this article is distinguished by the introduction of a wider range of patterns supporting the specification and verification of control flow, employed resources, and timed compliance constraints. In addition, compliance checking is not considered in [43]; therefore, the approach's applicability on compliance checking of business process models is unstudied.

Since the main focus of these approaches is not necessarily on compliance management, it is unclear which forms of compliance requirements (i.e. regarding control flow, data validation, employed resources, and real time) are supported. In addition, their support for the two compliance-related concepts—non-monotonicity and compensations are inherently not considered.

2.4 Summary and evaluation of related-work approaches

Table 1 presents a summary of the comparison between the key approaches that are discussed in the sections above.

Table 1 Summary and evaluation of related work

Study	Comp. req. abstract specification	Compliance requirements formal lang.	BP specification	BP abstraction model	Structural BP support	Verification approach	Bottom-up/top-down	Non-Mon.	Feedback
Abouzaid and Mullins [14]	–	π-logic	BPEL v2.0	BP-calculus	Control flow, data	Model checking	Top-down	No	No
Liu et al. [17]	Graphical BPSL	LTL	BPEL v2.0	π-calculus	Control flow, data	Model checking	Bottom-up	No	Yes
Giblin et al. [11]	–	Real-time temporal object logic	–	–	Control flow, time	Model checking	–	No	No
Awad et al. [15]	–	LTL	BPMN	FSA	Control flow, data	Process-mining techniques	Top-down	No	No
Yu et al. [16]	–	Temporal business rules	BPEL	Finite state automata of the rules	Control flow	Path finding, branching structure identification	Top-down	No	No
Awad et al. [18]	BPMN-Q	PLTL, CTL	BPMN	FSA	Control flow, data	Model checking	Bottom-up	No	Yes
Awad [44]	BPMN-Q	Query graph (labelled directed graph)	BPMN	Process graph (labelled directed graph)	Control flow	Graph matching	Top-down	No	No
Pelliccione et al. [23]	Property Sequence Chart	LTL	UML diagrams	FSA	Control flow	Model checking	Bottom-up	No	No
Governatori et al. [34]	–	FCL	– (natural-language contract)	Execution traces	Control flow, data, employed resources, time	Idealness approach	Bottom-up	Yes	No
Sadiq et al. [7]	–	FCL	BPMN	Execution traces	Control flow, data, employed resources, time	Idealness approach	Bottom-up	Yes	No
Governatori and Rotolo [35]	–	Temporalized violation logic	?	Execution traces	Control flow, data, employed resources, time	Implementation specific technique (theory compliance)	Bottom-up	Yes	No
Palmirani et al. [37]	–	LKIF rule language	?	?	Control flow, data, employed resources, time	?	Top-down	Yes	No
Goedertier and Vanthienen [6]	–	PENELOPE	BPMN	BP state space	Control flow, time	Implementation-specific technique	Top-down	No	No
Beeri et al. [41]	BP-QL	Directed labelled graph (BP patterns)	BPEL	Directed labelled graph	Control flow, data	Graph homomorphism	Top-down	No	No

'Comp. req. abstract specification' column in Table 1 contains the abstract/visual language used to represent compliance requirements (if any). 'Structural BP support' column indicates which BP structural facet the approach supports, i.e. control flow, data, employed resources, and/or temporal. 'Non-Mon.' column indicates if the approach provides a support to the handling of non-monotonic requirements. '?' in table indicates that it is unclear if the approach supports the relevant criteria.

3 Overview of the design-time business process compliance management

Figure 1 depicts an overview of the key practices and components of our design-time business process compliance management approach, and highlights the parts that outline the *scope* of this paper. The approach depicted in the figure is a concretized and detailed version of the generic approach that we previously introduced in [45].

There are two primary abstract roles involved in this approach: (i) a *business expert,* who is responsible for defining and managing business processes in an organization while taking compliance constraints into account, and (ii) a *compliance expert,* who is responsible for refining, internalizing, specifying, and managing compliance requirements stemming from external and internal sources in close collaboration with the business expert.

The approach encompasses two logical repositories: the *business process repository* and the *compliance requirements repository*, which may reside in a shared environment. Process models including service descriptions are defined and maintained in the BP repository, while compliance requirements and all relevant concepts are defined, maintained, and organized in the compliance requirements repository. These repositories foster the reusability of business and compliance specifications. We assume that these two specifications (BPs and compliance requirements) share the same constructs—mainly BP elements residing in the BP repository.

The BP definition (the right-hand side of Fig. 1) involves the specification of process models using the Business Process Execution Language (BPEL). However, as BPEL specifications are not grounded on a formal model, they should be transformed into some formal representation to enable their automated verification and analysis against formally specified compliance rules. The automated mapping of process specifications into a formal representation have been intensively studied in the literature (e.g. [11,14,17,46]). For this transformation, we have adopted and integrated the mapping framework proposed in [46] (and its prototypical implementation [47]). We have specifically chosen to exploit this approach due to its support to handle rich data manipulations via XPath expressions. This allows the analysis and validation of data exchanged as messages between participat-

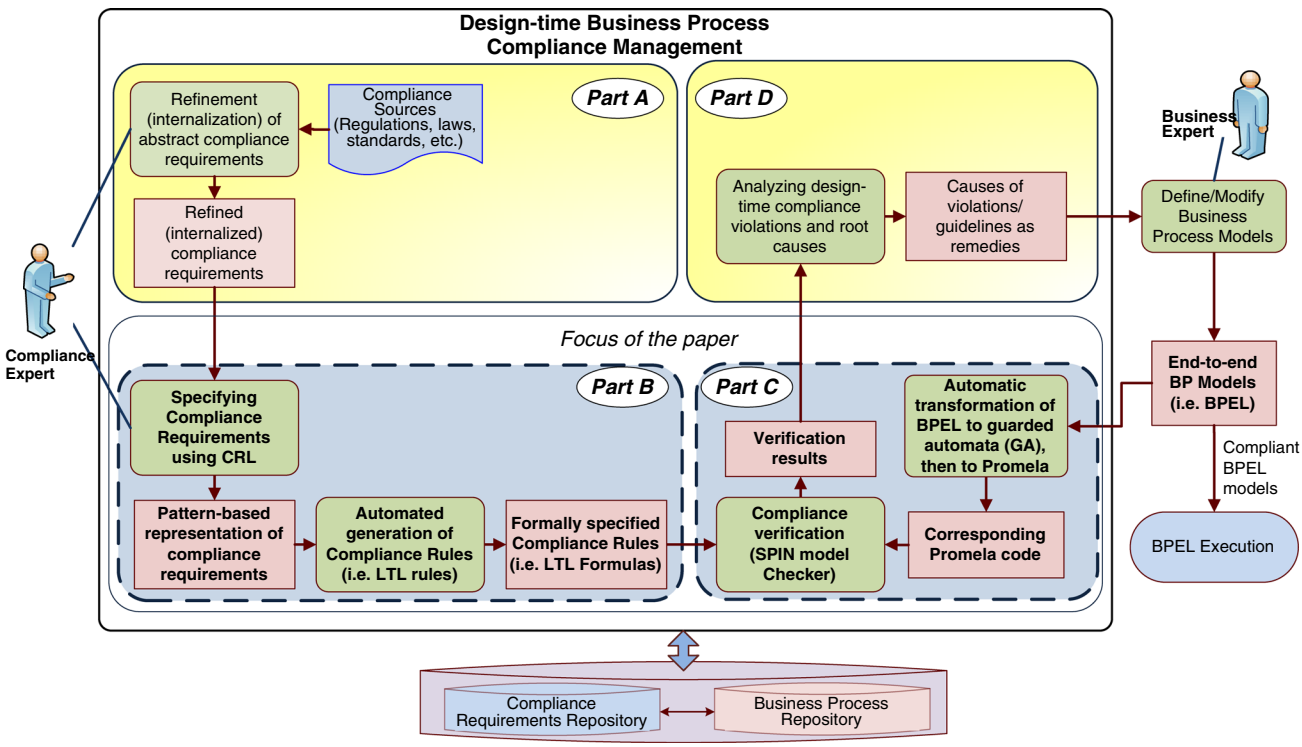

Fig. 1 Design-time business process compliance management approach

ing services. Following this approach, a BPEL specification is first mapped into an intermediate representation (guarded automata—GA), and then to *Promela* code—the verification language accepted by SPIN model checker [8] (Part C in Fig. 1). A brief description of this BPEL mapping is presented in Sect. 6.2.

On the other side (left-hand side in Fig. 1), compliance management practices commence with the *refinement* of compliance constraints originating from various compliance sources into a set of organization-specific compliance requirements (Part A in Fig. 1). This involves not only compliance but also business process domain knowledge. Our work on this part is presented in detail in [48] and [49]. The proposed refinement approach is based on the COSO [50] framework, which is the de facto framework used for establishing efficient internal control systems in organizations. The refinement approach briefly involves the following major steps:

I. Identification of the objectives and the abstract requirements enforced by compliance sources with which the organization has/agrees on to comply with.
II. Performing 'risk assessment' to identify the risks to the achievement of these objectives/abstract requirements imposed by the identified compliance sources.
III. Identifying, designing, and implementing 'controls' to mitigate the identified risks. Controls are concrete and organization-specific norms to be verified, enforced, or tested in order to ensure that compliance requirements are satisfied.
IV. Specifying formal compliance rules for those controls that can be formally represented and be used for automated process verification at design time and later phases.

For a more detailed discussion about this refinement methodology and its application on two real-life case studies, we refer the reader to [48] and [49]. To address the fourth step in the refinement methodology, compliance expert may apply and combine *compliance patterns* using the proposed high-level pattern-based Compliance Request Language (CRL) to render organization-specific compliance requirements (Part B in Fig. 1). This serves as an auxiliary step to represent internalized compliance requirements into formal statements (as LTL formulas for our case). These CRL expressions are then automatically transformed into LTL formulas. The verification of business process specifications (Part C in Fig. 1) mainly involves checking formal business process specifications (i.e. Promela code) against formal compliance rules (LTL rules) using the SPIN model checker [8]. SPIN is a popular open source software tool that is intensively used in both academia and industry for the formal verification of large-scale distributed software and hardware systems. The expected inputs to SPIN are as follows: a Promela code that captures the behaviour of the BPEL specification; and a set of LTL rules capturing relevant compliance requirements. The outcome of SPIN is a 'yes-no' answer indicating whether each LTL rule is satisfied or violated. In case of violations, the root-cause analysis approach that we have proposed in [51] and [52] may be conducted to analyse and reason about the root causes of design-time compliance violations and provide the user with suggestive guidelines of how to resolve the compliance anomalies. The business experts then alter the process specifications taking these guidelines into consideration, which is followed by the automated re-mapping of the BPEL specification into their formal forms (GA and Promela) and re-verifying against the set of applicable formal compliance rules. This process iterates until all violations are resolved and a statically compliant business process model is produced.

The parts in Fig. 1 enclosed with dotted lines and tagged as 'Part B' and 'Part C' illustrates the main focus of this paper. In particular, it concerns with the high-level representation of the refined compliance requirements as compliance patterns using CRL, the automatic mapping of CRL expressions into LTL formulas, and automatically verifying the resultant formal rules against BP models for design-time compliance assurance.

4 Case studies

In this section, we introduce two case studies that were conducted within the scope of the EU funded research project (http://www.compas-ict.eu). The case studies were performed in companies operating in different industry sectors and covered processes taking place in the e-business and banking domains. Taking into account the demands for strong regulation compliance schemes, such as Sarbanes-Oxley (SOX), ISO 27000 and sometimes contradictory needs of the different stakeholders, such business environments raise several interesting compliance requirements.

The *first* case study involved an Internet reseller company that offers products through online systems. The study covered a wide range of BPs, such as order processing, invoicing, cash receipting, and delivery. The *second* case study covered 'loan origination and approval' process that takes place in the banking domain. We use the second case study as a *running example* throughout this paper. The conducts and the overall findings of both studies are discussed later in Sect. 7.

A simplified model for the loan origination and approval process is depicted in Fig. 2 using BPMN (Business Process Model and Notation). BPMN is used in this section for illustration and presentation purposes; however, we consider BPEL for process specifications. The process flow can be described as follows: Once a customer loan request is received, the *credit broker* checks customer's banking privi-

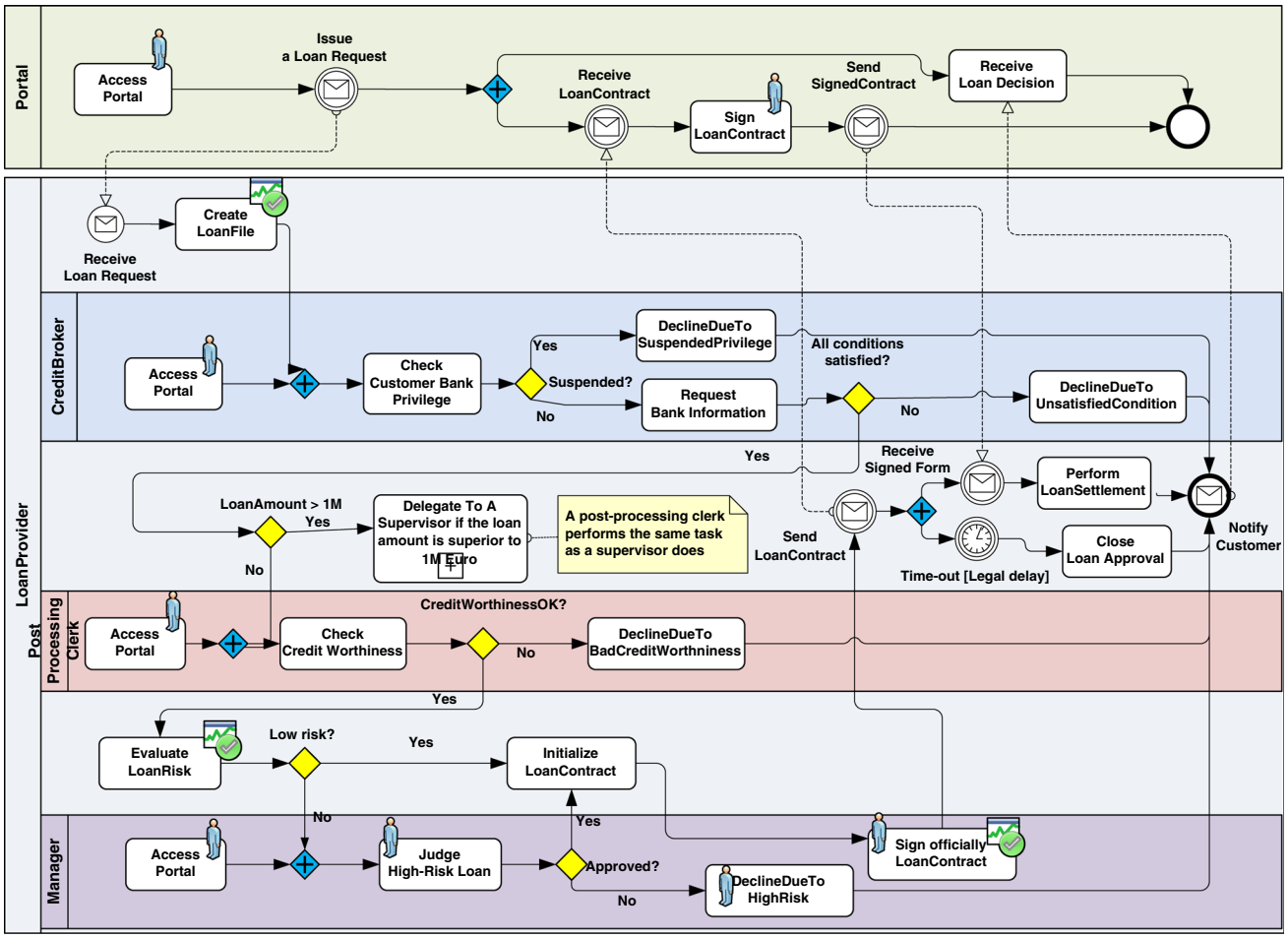

Fig. 2 BPMN model of the loan origination and approval process (simplified)

leges status. If privileges are not suspended, the credit broker accesses the customer information and checks whether all loan conditions are satisfied. Next, a loan threshold is calculated, and if the threshold amount is less than 1M Euros, the *post-processing clerk* checks the credit worthiness of the customer by outsourcing to a credit bureau service. Next, the *post-processing clerk* initializes the loan form and approves the loan. If the threshold amount is greater than 1M Euros, the *supervisor* is responsible for performing the same activities instead of the *post-processing clerk*. Next, the manager evaluates the loan risk, after which she normally signs the loan form and sends the form to the customer to sign. A legal waiting time of 7 days is provided to the customer to send back the signed contract. If a timeout occurs, which means that 7 days have passed and the customer has not sent the signed contract, the relevant loan approval application is closed by the system and the process terminates.

Table 2 shows an excerpt of the compliance requirements imposed on this loan approval scenario. This table is populated after applying our refined methodology [48,49] we summarized in Sect. 3 (identified potential risks and other related concepts are omitted for simplicity). The first and second columns of the table allocate a unique reference and an organization-specific interpretation of the requirement, respectively. The third column gives the high-level and abstract compliance requirements as they appear in the compliance sources (such as laws and regulatory documents), from which they originate. Finally, the fourth column refers to the associated compliance sources.

5 Compliance request language

In [53], we have analysed a wide range of compliance legislations and frameworks, including Sarbanes-Oxley [1], Basel III [2], IFRS [54], FINRA (NASD/SEC) [55], COSO [50], COBIT [56], and OCEG [57], and examined a variety of relevant works on the specification of associated compliance requirements. Based on this analysis and our joint work with two industrial companies (*PriceWaterHouseCoopers—PwC*, the Netherlands and *Thales Services*, France), we have iteratively and incrementally identified structural patterns of

Table 2 An excerpt of the compliance requirements relevant to the case scenario

ID	Refined compliance requirement	(High-level) compliance requirement	Compliance sources
Req.1	The customer should receive an automated email notification when his personal data is collected by the 'credit bureau service'	Customer's personal data should be handled confidentially	95/46/EC (data protection directive)
Req.2	The checking of the customer bank privilege that is followed by checking of her credit worthiness must take place before determining the risk level of the loan application	Loan should be granted with adequate level of assurance	Internal bank policy
Req.3	The activity 'customer bank privilege check' (to be performed by credit broker or supervisor) should be segregated from 'credit worthiness check' (to be performed by post-processing clerk)	Duties in loan processing should be adequately segregated	Sarbanes-Oxley Sec. 404 - ISO 27002-10.1.3
Req.4	The branch office manager checks whether risks are acceptable and makes either the final approval or rejection of the request	Duties in loan processing should be adequately segregated	Sarbanes-Oxley Sec. 404 - ISO 27002-10.1.3
Req.5	The offer in the signed loan contract is valid for 7 working days and afterwards it is closed	Bank offers for customers and third parties should be valid for certain time periods	Internal bank policy
Req.6	If the loan request's credit exceeds 1 million Euro ($1M$ €) the clerk supervisor checks the credit worthiness of the customer. The lack of the supervisor check immediately creates a suspense file. In case of failure of the creation of a suspense file, the manager is notified by the system	Duties in loan processing should be adequately segregated	Sarbanes-Oxley Sec. 404 - ISO 27002-10.1.3
Req.7	Checking banking privileges is optional for trusted (gold) customers. If a trusted (gold) customer's loan request is less than 1M Euros, the evaluation of the loan risk is not performed	Customers can take advantage of special treatments with respect to the customer categorization policy	Internal bank policy

frequently recurring (compliance) requirements imposed on business processes. Based on the findings of the analysis, we have also identified a set of features a formal language for compliance requirements should possess, which is reported in detail in [58] and [59]. This paper builds on this work by providing an abstract pattern-based language (CRL) that spans over three structural aspects (out of four) of compliance requirements (control flow, employed resources, and temporal) and addresses the *usability* concern of temporal logic. In addition, it supports the specification and verification of *non-monotonic* requirements and *compensations*.

Our analysis also revealed different categorizations of compliance requirements. Requirements can be applicable to controls (checks) that pursue either a *preventive* or *detective* approach. Preventive controls help to keep violations from occurring. Examples include authorizations, segregation of duties, and supervisory approval. Detective controls, on the other hand, often produce information regarding an occurred violation to help understand its causes. Examples are management reviews and reconciliations.

With respect to the instruments used for the control, the compliance requirements can also be categorized as *process*, *technical*, and *physical* [57]. *Process* -related requirements are relevant to the policies and practices concerning the design and execution of BPs. Authorizations, approvals, inspections, segregation of duties applied through business tasks, and other elements are examples of such requirements. *Technical* requirements involve the use of devices or systems mainly for authentication, encryption, or security purposes. Examples include firewalls and intrusion prevention/detection systems. *Physical* requirements involve largely the institution of physical means, such as locks, fences, and alarms, to guard critical assets. In identifying the patterns, we focused on *preventive-process controls* that can be applied for automated design-time verification. The compliance patterns studied in this paper are not applicable or effective for representing the compliance requirements that are applied following a *detective* approach, or those that are classified as *technical* or *physical*.

5.1 CRL meta-model

Figure 3 presents the meta-model of the Compliance Request Language (CRL) represented by UML class diagram. The *compliance pattern* class is the core element of the language, and each pattern is a sub-type of it. A *compliance pattern* is a high-level domain-specific template used to represent frequently occurring compliance constraints. The compliance pattern class is sub-divided into four main sub-classes of patterns, namely *atomic*, *resource*, *composite*, and *timed*.

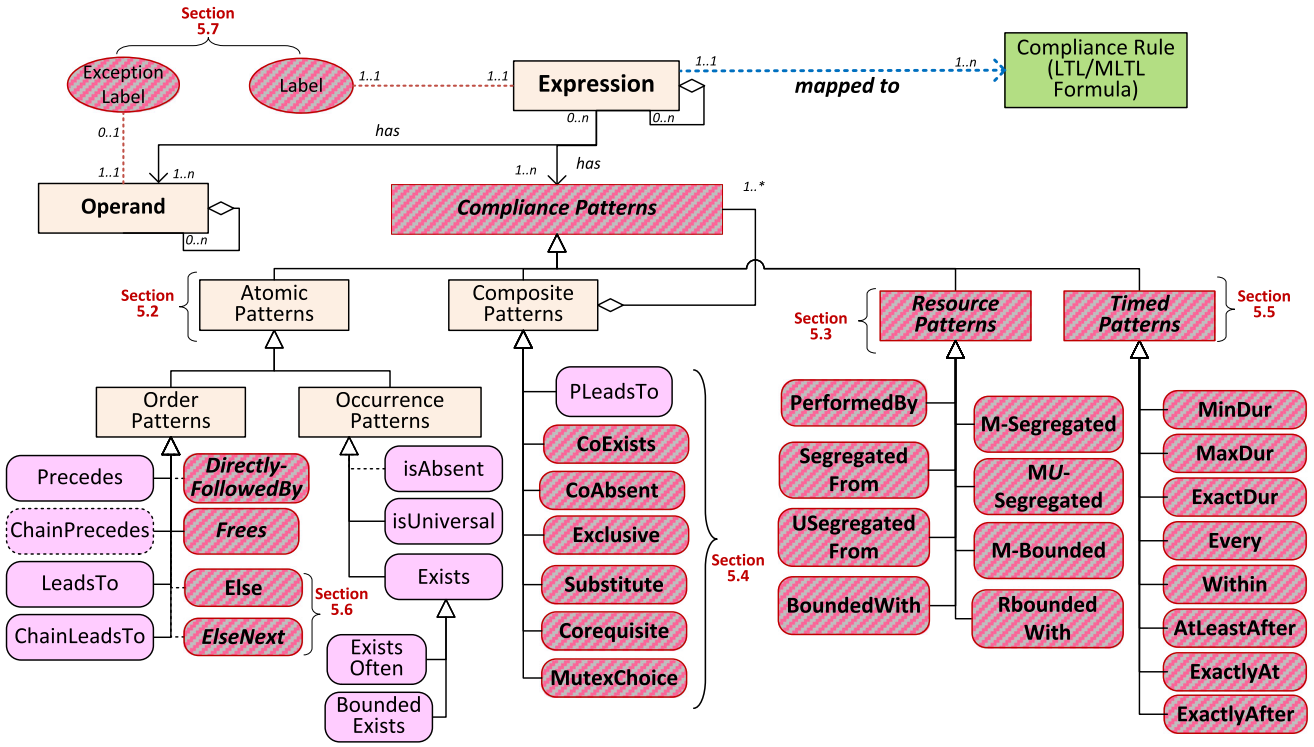

Fig. 3 Compliance request language meta-model

Atomic patterns deal with occurrence and ordering constraints. Some patterns (those 'non-shaded' in Fig. 3) are adopted from Dwyer's property specification patterns [22]. *Resource patterns* capture recurring requirements related to task assignments and authorizations, such as segregation of duties.*Composite patterns* are built up from combinations (nesting) of multiple atomic patterns via Boolean operators to allow for the definition of complex requirements. *Timed patterns* are used in combination with other compliance patterns to capture time-dependent constraints. We elaborate these types of patterns in Sects. 5.2 through 5.6.

As shown in Fig. 3, an *expression* mainly comprises *compliance patterns* (*patterns* in short) and *operands*. Expressions can combine multiple (sub-)expressions by using Boolean operators. For example, the expression *[(P Precedes Q) And (R Exists)]* comprises two sub-expressions, where:

Precedes and *Exists* are atomic patterns.

P, Q, and R are operands (that typically represent BP elements).

'And' represents the conjunction Boolean operator.

Operands take the form of BP elements (such as activities, events, business objects), their attributes, or conditions on them. For example, the expression '*CreateOrder LeadsTo ApproveOrder*' has two operands (activities in this case) connected via the *LeadsTo* atomic pattern. Similar to expressions, operands can also be combined and nested via Boolean operators, e.g.:

$$(CheckCustomerPrivilege\ And\ (Loan.Amount > 1M\text{'€'}))\ LeadsTo\ (CheckCreditWorthiness.Role('Supervisor'))$$

An expression built from compliance patterns and operands has a direct mapping to LTL formulas. The formal description of the CRL grammar defining its syntax can be found in [60]. CRL has formally defined operational semantics given by the mapping into LTL formulas [61]. In the following, we describe compliance patterns classes in more detail by exemplifying them using the running scenario introduced in Sect. 4.

5.2 Atomic patterns

Atomic patterns can be used to describe the requirements that involve basic occurrence and ordering of BP elements. They are founded on Dwyer's property specification patterns [22]. We extended Dwyer's patterns with four *atomic* patterns: *Else, ElseNext, DirectlyFollowedBy* ,and *Frees*. The *Else* and *ElseNext* atomic patterns are used to represent compensations in a way analogous to *If-then-else* statements. These two patterns are discussed in detail in a *separate* section (Sect. 5.6). Atomic patterns can be summarized as follows (given P and Q as operands):

Occurrence patterns

- *P isAbsent* indicates that P should never hold throughout the BP model.
- *P isUniversal* indicates that P should always hold throughout the BP model.
- *P Exists* mandates that P must hold at least once within the BP model.
- *P BoundedExists* (atLeast/Exactly/atMost) k indicates that P must hold at least/exactly/at most k times, respectively, within the BP model.
- *P ExistsOften* indicates that P must hold more than once within the BP model.

Order patterns

- *P Precedes Q* indicates that Q must always be preceded by P.
- *P LeadsTo Q* indicates that P must always be followed by Q.
- $(P_1, \ldots, P_n)$ *ChainPrecedes* $(Q_1, \ldots, Q_m)$ indicates that the sequence $Q_1, \ldots, Q_m$ must be preceded by the sequence of $P_1, \ldots, P_n$.
- $(P_1, \ldots, P_n)$ *ChainLeadsTo* $(Q_1, \ldots, Q_m)$ indicates that the sequence $P_1, \ldots, P_n$ must be followed by the sequence $Q_1, \ldots, Q_m$.
- *P DirectlyFollowedBy Q* represents a strict case of the *LeadsTo* pattern, which requires P to be directly followed by Q.
- *P Frees Q* indicates that the second operand Q has to be true until and including the point where the first operand P first becomes true. For example, a requirement stated as: 'the status of the loan request should always be *pending* until the check of the loan risk returns a positive response', can be specified in CRL as:

$$Loan.Approved = \text{`}Yes\text{'}\ Frees\ Loan.Status = \text{`}Pending\text{'}.$$

Atomic patterns exemplified:

Returning back to our running example introduced in Sect. 4, we can represent requirements *Req.1* and *Req.2* given in Table 2 in CRL as follows:

Req.1: The customer should receive an automated email notification when his personal data is collected by the 'Credit Bureau service'

R1: ((RequestBankInformation Or CheckCredit Worthiness) $DirectlyFollowedBy$ NotifyCustomer)

Regarding Req.1, customer information is collected by conducting a credit bureau service to check the credit worthiness of the customer. In case the loan requester is already a customer of the bank, the credit broker can directly access her personal information from the bank database by invoking 'request bank information' activity (as shown in Fig. 2). The pattern expression $R1$ ensures that the customer will be notified immediately after her data has been accessed.

Req.2: The checking of the customer bank privilege that is followed by checking of her credit worthiness must take place before determining the risk level of the loan application

R2: $((CheckCustomerBankPrivilege, CheckCreditWorthiness)ChainPrecedes\ EvaluateLoanRisk)$

$R2$ uses the *ChainPrecedes* pattern of Dwyer's property specification patterns [22], which mandates the sequence of *CheckCustomerBankPrivilege* and *CheckCreditWorthiness* to precede *EvaluateLoanRisk* activity.

5.2.1 From atomic patterns to LTL

LTL [61] is a logic used to formally specify 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 the system. The formulas take the form Af, 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 of LTL formulas are as follows [61]:

- $\top$ and $\bot$ are formulas (where $\top$ represents tautology and $\bot$ represents contradiction).
- 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 $\neg f$, $f \vee g$, $f \wedge g$, Xf, Ff, Gf, fUg, fWg, and fRg are path formulas (where '$\vee$' represents 'or' and '$\wedge$' represents 'and' logical operators), such that:
  - *G (always)* indicates that formula f must be true in all the states of the path.
  - *X (next time)* indicates that the formula f must be true in the next state of the path.
  - *F (eventually)* indicates that formula f will be true at some state in the future.
  - *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.
  - *W (weak until)* represents the same semantics as *until*; however, it is evaluated to true if the second formula g never occurs.

Table 3 Mapping rules from atomic patterns to LTL

Atomic pattern	Pattern-based expression	Description	LTL representation
isAbsent	P isAbsent	P should not exist throughout the BP model	$G(\neg P)$
Exists	P Exists	P should occur at least once within the BP model	$F(P)$
Bounded-Exists	P BoundedExists ≤ 2 *with bound ≤ 2	P must occur at most 2 times within the BP model	$\neg PW(PW \quad (\neg PW(PW\neg F(P))))$
	P BoundedExists≥ 2 *with bound ≥ 2	P must occur at least 2 times within the BP model	$\neg PW(PW \quad (\neg PW(P)))$
isUniversal	P isUniversal	P should always be true throughout the BP model	$G(P)$
Precedes	P Precedes Q	Q must always be preceded by P	$\neg QWP$
Chain-Precedes	P Precedes (S,T)	A sequence of S, T must be preceded by P.	$(F(S \wedge XF(T))) \rightarrow ((\neg S)UP))$
	(S,T) Precedes P	P must be preceded by a sequence of S, T	$F(P) \rightarrow (\neg PU(S \wedge \neg P \wedge X(\neg PUT)))$
LeadsTo	P LeadsTo Q	P must always be followed by Q	$G(P \rightarrow F(Q))$
Chain-LeadsTo	P LeadsTo (S,T)	P must be followed by a sequence of S, T	$G(P \rightarrow F(S \wedge XF(T)))$
	(S,T) LeadsTo P	A sequence of S, T must be followed P	$G(S \wedge XF(T) \rightarrow X(F(T \bigwedge F(P))))$
Exists-Often	P ExistsOften	P must occur frequently within the BP model	$GF(P)$
DirectlyFollowedBy	P DirectlyFollowedBy Q	Requires P to be directly followed by Q	$G(P \rightarrow X(Q))$
Frees	P Frees Q	The second operand Q has to be true until and including the point where the P first becomes true	PRQ

P, Q, S and R are operands that indicate BP elements (such as activities, roles, data objects, events), their attributes, or conditions on them. The mapping rules for Dwyer patterns (see Fig. 3 non-shaded patterns) are based on [26]. The mapping for the ExistsOften pattern is introduced in [28]

- *R (release)* indicates that the second formula g has to be true until and including the point where the first formula f first becomes true; if f never becomes true, g must remain true forever. *R (release)* is the dual of *U (Until)*.

The mapping scheme from a compliance pattern into LTL enables the automated transformation of CRL expressions into a set of LTL formulas. Table 3 lists the mapping rules to transit from atomic patterns into LTL.

Applying the mapping rules given in Table 3 automatically generates the LTL formulas that correspond to the CRL expressions $R1$ and $R2$ given above. The generated LTL formulas of $R1$ and $R2$ are:

$$\boldsymbol{R1'}: G((RequestBankInformation \bigvee CheckCreditWorthness) \rightarrow X(NotifyCustomer))$$

$$\boldsymbol{R2'}: F(EvaluateLoanRisk \rightarrow \neg EvaluateLoanRisk\, U(CheckCustomerBankPrivilege \wedge \neg EvaluateLoanRisk \wedge X(\neg EvaluateLoanRisk\, U\, CheckCreditWorthiness)))$$

5.3 Resource patterns

'Employed resources' involves mainly the task allocations, access control, and authorization constraints, and constitutes one of the important structural facets of BP compliance. CRL addresses this dimension through the *resource patterns*, which typically involve some basic BP concepts, in particular *role*, *user*, and *task (or BP activity)*. We assume that tasks are assigned to roles and users perform the tasks through the roles they are assigned to. As shown in Fig. 3, we introduce eight resource patterns that are described in Table 4.

Compliance requirements on employed resources typically require run-time information to be verified. That is, many of such requirements can only be partially verified for compliance at design time. Although the main focus of this paper is on design-time compliance, our underlying motivation is to establish a *preventive* viewpoint in compliance management. This requires efficient integration of design-time and run-time verification techniques. Accordingly, we found it meaningful to discover and propose patterns that can also be used to specify requirements that may only be checked at the subsequent run-time monitoring phase. In that respect, we mark the rules involving *users* to be checked and monitored at run-time, since the assignment of specific users to particular roles might not be known until such information is available

Table 4 Resource pattern descriptions and their mapping rules into LTL

Resource pattern	Description	LTL mapping rule
t PerformedBy R	No other role than R is allowed to perform activity t	$G(t \rightarrow t.Role(R))$
t_1 SegregatedFrom t_2	Activities t_1 and t_2 must be performed by different roles and users	$G\,(t_1.Role\,(R) \rightarrow \neg\,(t_2.Role\,(R))$ $\wedge\, G\,(t_1.User\,(U) \rightarrow \neg(t_2.User\,(U))$
t_1 USegregatedFrom t_2	Activities t_1 and t_2 must be performed by different users	$G\,(t_1.User\,(U) \rightarrow \neg(t_2.User\,(U))$
t_1 BoundedWith t_2	Activities t_1 and t_2 must be performed by the same user	$G(t_1.User(U_1) \rightarrow t_2.User(U_1)) \wedge G(t_2.User(U_1)$ $\rightarrow t_1.User(U_1))$
(t_1, t_2) RBoundedWith R_1	Activities t_1 and t_2 must be performed by the same role R_1 but different users	$G\,(t_1 \rightarrow t_1.Role\,(R_1)) \wedge G\,(t_2 \rightarrow t_2.Role\,(R_1))$ $\bigwedge G\neg\,(t_1.User\,(U_1) \wedge t_2.User\,(U_1))$ $\bigwedge G\neg\,(t_1.User\,(U_2) \wedge t_2.User\,(U_2))$ $\wedge[U_1, U_2] \in GetUser(R_1)$
$(t_1, t_2, \ldots, t_n)$ M-Segregated $(R_1, R_2, \ldots, R_m)$	Roles $(R_1, R_2, \ldots, R_m)$ should be involved in the performance of activities $(t_1, t_2, \ldots, t_n)$	$G(\wedge_{1\le i\le n}(t_i \rightarrow \vee_{1\le j\le m}(t_i.Role\,(R_j))))$ $\wedge_{1\le k\le m}$ $(G(\neg(\wedge_{1\le l\le n}(t_l.Role(R_k)))))$ Such that: $n, m \in \aleph$ are the number of tasks and involved roles, respectively, $n \ge m$ and $i, j, k, l \in \aleph$
$(t_1, t_2, \ldots, t_n)$ M-USegregated $(U_1, U_2, \ldots, U_h)$	Indicates that users $(U_1, U_2, \ldots, U_h)$ should be involved in the performance of activities $(t_1, t_2, \ldots, t_n)$	$G(\wedge_{1\le i\le n}(t_i \rightarrow \vee_{1\le j\le h}(t_i.User\,(U_j))))$ $\wedge_{1\le k\le h}$ $(G(\neg(\wedge_{1\le l\le n}(t_l.User(U_k))))$ Such that: $n, h \in \aleph$ are the number of tasks and involved users, respectively, $n \ge h$, and $i, j, k, l \in \aleph$
$(t_1, t_2, \ldots, t_n)$ M-Bounded U_1	Indicates that activities $(t_1, t_2, \ldots, t_n)$ must all be performed by the same user U_1	$G(\wedge_{1\le i\le n}(t_i.User(U_1)))$ Such that $n \in \aleph$ is the number of tasks and $i \in \aleph$

at run-time. This applies to rules that are generated by using the following resource patterns: *SegregatedFrom, USegregatedFrom, BoundedWith, RBoundedWith MU-Segregated*, and *M-Bounded*.

To give an example, consider the mapping rule of the *SegregatedFrom* pattern as given in Table 4. It is comprised of two rules, i.e.

- $f_1 : G\,(t_1.Role\,(R) \rightarrow \neg(t_2.Role\,(R))$: This LTL rule ensures that activities t_1 and t_2 are performed by different roles.
- $f_2 : G\,(t_1.User\,(U) \rightarrow \neg(t_2.User\,(U))$: This LTL ensures that that activities t_1 and t_2 are performed by different users.

Rule f_1 can be verified during design time, while rule f_2 is generated and marked for subsequent run-time monitoring due to the absence of this specific contextual information during design time.

Resource patterns exemplified

Compliance requirements Req.3 of the running scenario as described in Table 2 represents a resource allocation and authorization constraint that can be represented in CRL as follows:

Req.3: The activity 'Customer bank privilege check' (to be performed by Credit Broker) should be segregated from 'credit worthiness check' (to be performed by post-processing clerk or supervisor)

***R*3.1**: (CheckCustomerBankPrivilege $PerformedBy$ 'CreditBroker')
***R*3.2**: (CheckCreditWorthiness $PerformedBy$ 'PostProcessingClerk' Or 'Supervisior')
***R*3.3**: ('CheckCustomerBankPrivilege' $SegregatedFrom$ CheckCreditWorthiness)

Req.3 represents the typical segregation of duties compliance requirement. First, *R*3.1 and *R*3.2 ensure that *CheckCustomerBankPrivilege* and *CheckCreditWorthiness* activities are assigned to the appropriate personnel. Then, *R*3.3 checks if these two activities are adequately segregated.

Applying the mapping rules given in Table 4, the LTL formulas that correspond to the pattern expressions *R3.1 to R3.3* can be automatically generated as follows:

R3.1′: $G(CheckCustomerBankPrivilege \rightarrow (CheckCreditWorthiness.Role(`CreditBroker'')))$
R3.2′: $G(CheckCreditWorthiness \rightarrow (CheckCreditWorthiness.Role(`PostProcessingClerk') \vee CheckCreditWorthiness.Role(`Supervisior')))$
R3.3′: $G((CheckCustomerBankPrivilege.Role(R) \rightarrow G(\neg(CheckCreditWorthiness.Role(R))$
R3.4′: $G((CheckCustomerBankPrivilege.User(U) \rightarrow G(\neg(CheckCreditWorthiness.User(U))$

As discussed above, the generated LTL rules $R3.1'$, $R3.2'$, and $R3.3'$ are checked during design time, while rule $R3.4'$ is reserved for run-time monitoring.

5.4 Composite patterns

To facilitate the definition of more complex requirements, composite patterns utilize Boolean logical operators (*Not*, *And*, *Or*, *Xor*, *Imply,* and *Iff*) to enable the nesting of multiple patterns. For example, *PLeadsTo* pattern is an '*And*' composition of '*P Precedes Q*' *And* '*P LeadsTo Q*' [21], which indicates that operands *P* and *Q* should 'occur' and must take place sequentially. Table 5 presents details regarding the CRL composite patterns.

Composite patterns exemplified

Compliance requirement *Req.4* of the running scenario (Table 2) forms a combination of task allocation and composite compliance requirement. It can be represented in CRL as follows:

Req.4: *The branch office Manager checks whether risks are acceptable and makes either the final approval or rejection of the request.*

R4.1: $(EvaluateLoanRisk.LowRisk = `No' LeadsTo\ JudgeHighRiskLoan)$
R4.2: $JudgeHighRiskLoan\ PerformedB\ `Manager'$
R4.3: $SignOfficiallyLoanContract\ PerformedBy\ `Manager'$
R4.4: $DeclineDueToHighRisk\ PerformedBy\ `Manager'$
R4.5: $(JudgeHighRiskLoan\ LeadsTo\ (SignOfficiallyLoanContract\ MutexChoice\ DeclineDueToHighRisk))$

Compliance requirement *Req.4* can be represented using five expressions. $R4.1$ checks if *JudgeHighRiskLoan* activity takes place. Then, expressions $R4.2$, $R4.3$, and $R4.4$ ascertain that corresponding activities are assigned to the proper roles. Finally, $R4.5$ checks whether *JudgeHighRiskLoan* is followed by either *SignOfficiallyLoanContract* or *DeclineDueToHighRisk* (but not both or neither of them).

Applying the mapping rules given in Tables 3, 4, and 5, the LTL formulas that correspond to the pattern expressions *R4.1 to R4.5* can be automatically generated as follows:

R4.1′: $G(EvaluateLoanRisk.LowRisk = `No' \rightarrow F(JudgeHighRiskLoan))$
R4.2′: $G(JudgeHighRiskLoan \rightarrow JudgeHighRiskLoan.Role(`Manager'))$
R4.3′: $G(SignOfficiallyLoanContract \rightarrow SignOfficiallyLoanContract.Role(`Manager'))$
R4.4′: $G(DeclineDueToHighRisk \rightarrow DeclineDueToHighRisk.Role(`Manager'))$
R4.5′: $G(JudgeHighRiskLoan \rightarrow (F((SignOfficiallyLoanContract \wedge G(\neg DeclineDueToHighRisk))) F((DeclineDueToHighRisk \wedge G(\neg SignOfficiallyLoanContract)))))$

5.5 Timed patterns

Time dimension is another key aspect in BP compliance and CRL addresses time-related requirements with eight patterns. Timed patterns should be used in conjunction with other compliance patterns (atomic or composite patterns) forming a 'timed composite pattern' expression. However, not every timed pattern can be composed with all compliance patterns. In total, we defined 51 possible combinations, from which a subset is presented in Table 6. For the complete list of combinations, the reader is referred to [60].

Regarding the mapping from timed patterns to corresponding formal statements, LTL lacks the support to such requirements. Various extensions to LTL, such as metrical temporal logic (MTL) and ForSpec temporal logic (FTL) [62] have been proposed in the literature to overcome this limitation.

We have selected MTL for this purpose as MTL extends LTL and thus holds the same semantics (and formation rules). MTL is interpreted over a discrete time domain (over the set of natural numbers $\aleph$). Its temporal operators can be annotated with a real-time expression I that represents a specific time interval. For example, $F_{\geq 5}\emptyset$ represents that in some future state after at least a delay of 5 time units, $\emptyset$ must hold. MTL uses the digital-clock model [63], such that an external, discrete clock progresses at a fixed rate. The granularity of the time can be set by the user.

Similar to the case in resource patterns, many of the rules generated using timed patterns can be fully checked only at

Table 5 Mapping rules from composite patterns into LTL

Composite pattern	Description	Atomic pattern equivalence	LTL representation
P CoExists Q	The presence of P mandates that Q is also present	$(PExists) \rightarrow QExists)$	$F(P) \rightarrow F(Q)$
P CoAbsent Q	The absence of P mandates that Q is also absent	$(P\,isabsent) \rightarrow (Q\,isabsent)$	$G(\neg P) \rightarrow G(\neg Q)$
P Exclusive Q	The presence of P mandates the absence of Q. And presence of Q mandates the absence of P	$((P\ Exists) \rightarrow (Q\,isabsent)) \bigwedge((Q\ Exists) \rightarrow (P\,isabsent))$	$(F\,(P) \rightarrow G(\neg Q)) \bigwedge (F\,(Q) \rightarrow G(\neg P))$
Q Substitute P	Q substitutes the absence of P	$(P\,isabsent) \rightarrow (Q\,exists)$	$G(\neg(P)) \rightarrow F(Q)$
P Corequisite Q	P and Q should either exist together or to be absent together	$(P\ Exists)\,iff\,(Q\ Exists) = ((P\ Exists) \rightarrow (Q\ Exists)) \bigwedge((Q\ Exists) \rightarrow (P\ Exists))$	$(F\,(P) \rightarrow F\,(Q)) \bigwedge (F\,(Q) \rightarrow F(P))$
P MutexChoice Q	Either P or Q exists but not any of them or both of them	$(P\,exists)\ Xor\ (Q\,exists) = \big((P\,exists) \bigwedge (Q\,isabsent)\big) \vee ((Q\,exists) \bigwedge (P\,isabsent))$	$\big(F\,(P) \bigwedge G\,(\neg\,(Q))\big) \vee (F(Q) \bigwedge G(\neg(P)))$

Table 6 Mapping rules from combinations of compliance/timed patterns into MTL

Timed pattern	Atomic/composite pattern	Timed composite pattern expression	Description	MTL representation
Every	Exists	P Exists Every k	Specifies the amount of time a BP element (e.g. activity, data object) has to hold at least once	$F_{\leq k}(P)$
Within	LeadsTo	P LeadsTo Q Within k	Indicates that BP element Q has to follow P within k time units after the occurrence of P	$G(P \rightarrow F_{\leq k}(Q))$
	Substitute	P Substitutes Q Within k	Q substitutes the absence of P within at most k time units from the start of the BP	$G(\neg P \rightarrow F_{\leq k}(Q))$
AtLeast After	LeadsTo	P LeadsTo Q AtLeastAfter k	Indicates that BP element Q has to follow P after k time units after the occurrence of P	$G(P \rightarrow F_{\geq k}(Q))$
	Precedes	P Precedes Q AtLeastAfter k	Indicates that BP element P should occur before each occurrence of Q. The time difference between P and Q should be more than or equals to k time units	$(\neg Q\,W\,P) \bigwedge (G_t\,\big(Q \rightarrow F_{\leq t-k}(P)\big))$
ExactlyAt	LeadsTo	P LeadsTo Q ExactlyAt k	Indicates that BP element Q has to follow P exactly at time k	$G(P \rightarrow F_{=k}(Q))$
	Release	P Frees Q ExactlyAt k	P must occur exactly at the elapse of k time units from the occurrence of Q to free it	$P\,R_{=k}\,Q$
Exactly After	LeadsTo	P LeadsTo Q ExactlyAfter k	Indicates that BP element Q has to follow P exactly after k time units from the occurrence of P	$G(P \rightarrow F_{=k}(X(Q)))$
	Inclusive	P Inclusive Q ExactlyAfter k	The presence of P mandates that Q is also present in the next state exactly after the elapse of k time units from the time of occurrence of P	$F_t(P \rightarrow F_{=\mathrm{t}+k}\,(X\,(Q)))$

run-time as we typically lack the time information at design time. Thus, respective compliance rules are reserved for subsequent run-time compliance monitoring. In some cases, however, the process specification can be enriched with time relevant information. *Timeouts* that are defined in BPEL models are of such kind. In these cases, verification of such rules at design time is also possible to make sure that the process is specified properly. Req.5 in Table 2 is an example of such a requirement.

Timed patterns exemplified:

Compliance requirement *Req.5* given in Table 2 can be represented in CRL as follows:

Req.5: *The offer in the signed loan contract is valid for 7 working days and afterwards it is closed*

***R*5.1**: $((SendSignedLoanContract\ LeadsTo\ RecieveCustomerSignedContract)\ Within\ 7)$
***R*5.2**: $((CloseLoanContract\ Substitutes\ RecieveCustomerSignedContract)\ ExactlyAfter\ 7)$

Req.5 can be represented by two expressions $R5.1$ and $R5.2$, which use the combination of *LeadsTo—Within* patterns, and *Substitutes—ExactlyAfter* patterns, respectively. $R5.1$ states that *SendSignedLoanContract* should always be followed by *RecieveCustomerSignedContract* within a time interval less than or equal to 7 time units (*days* in the running scenario) from the start of the first activity. The expression $R5.2$ indicates that *CloseLoanContract* substitutes the absence of *RecieveCustomerSignedContract* in the next state after the elapse of 7 days.

By applying the mapping rules given in Tables 3, 5, and 6, the LTL/MTL formulas that correspond to $R5.1$ and $R5.2$ are as follows:

***R*5.1′**: $G(SendSignedLoanContract \rightarrow F_{\leq 7}(RecieveCustomerSignedLoanContract))$
***R*5.2′**: $G(\neg RecieveCustomerSignedLoanContract \rightarrow F_{=7}(X(CloseLoanContract)))$

5.6 Capturing compensations with Else and ElseNext patterns

It is important to specify compensations to the violations of certain compliance requirements to handle *certain* exceptional situations. For example, a requirement may necessitate 'sending a confirmation message to the customer via e-mail after receiving her application'. A technical failure may prevent this task to be performed and lead to the violation of the requirement. In this case, it may be desirable to define compensation actions (for example, sending a confirmation via SMS) to repair this violation.

The compensation actions can be defined in the form of chains such that each action repairs the violation of its predecessor. This kind of requirements is analogous to *if-then-else* statements, which impose a prioritization on the order of evaluation of its conditions and actions. We call the action that appears just after the 'then' part of the if-then-else statement as the *primary action*, and every action after each 'else' part is called *compensation action*. If the primary action holds, none of its compensations would take place. Similarly, one and only one action from the primary action and compensation actions should hold. Such a requirement is considered violated if neither of its primary action nor compensation actions holds.

To enable the specification and verification of such requirements in CRL, we use *Else* and *ElseNext* atomic patterns in conjunction with *LeadsTo* and *DirectlyFollowedBy* atomic patterns (explained in Sect. 5.2). Accordingly, a compensable CRL expression can be formed as:

$$P\ (LeadsTo|DirectlyFollowedBy)\ P_1(Else|ElseNext)\ P_2 \ldots (Else|ElseNext)\ P_n$$

Where:

- P is the rule condition.
- P_1 is the primary action.
- $P_2 \ldots P_n$ are the compensation actions.

'$P\ (LeadsTo|DirectlyFollowedBy)\ P_1$' captures the 'if-then' part of the if-then-else statement. Accordingly, the primary action P_1 should take place *either* immediately after P holds (by the *DirectlyFollowedBy* pattern) *or* sometime eventually (by the *LeadsTo* pattern). To capture the 'else' part, either *ElseNext* or *Else* patterns are used, indicating that a compensation action should take place *either* immediately after its predecessor action (that fails to compensate the failure) or sometime eventually, respectively.

For example, the requirement *Req.6* given in Table 2 can be represented in CRL using *LeadsTo*, *Else*, and *ElseNext* patterns as follows:

Req.6: *If the loan request's credit exceeds 1 million Euro* ($1M$ €) *the Clerk Supervisor checks the credit worthiness of the customer. The lack of the supervisor check immediately creates a suspense file. In case of failure of the creation of a suspense file, the manager is notified by the system*

***R*6:** $Loan.Amount > \text{'}1M\text{'}\ LeadsTo\ CheckCreditWorthiness.Role(\text{'}Supervisor\text{'})\ ElseNext\ CreateSuspenseFile\ Else\ NotifyManager$

During design-time verification, the objective is to ensure that the sequence of the primary action and compensation actions is structurally encoded in the BP model, and there is a transition from one action (primary or compensation) to the next only if the first action could not be completed successfully. This necessitates checking the existence of a decision point after each action that checks whether the action is completed successfully or not. Thus, the mapping rule of a compensable expression for *design-time verification* can be defined as follows:

$P\ (LeadsTo|DirectlyFollowedBy)P_1(Else|\ ElseNext)$
$P_2 \ldots (Else|ElseNext)\ P_n$
$= G(p \rightarrow F|X(p1 \wedge_{1 \leq i < n-1} (F|X(p_i NotSucceed)$
$\wedge (p_i NotSucceed \rightarrow F|X(p_{i+1})))))$
Where:

- G, F, X represent 'always', 'eventually' and 'next' temporal operators, respectively;
- P represents the compensable rule condition;
- P_1 represents the primary action;
- $P_2, \ldots, P_n$ are the compensation actions;
- and $i, n \in \aleph$
- $p_i NotSucceed$ represents the decision point that checks whether p_i took place

Based on this mapping rule, the generated LTL formula from the expression $R6$ is as follows:

$\boldsymbol{R6'}$: $(G(Loan.Amount > \text{'}1M\text{'} \rightarrow F(checkCredit$
$Worthiness.Role(\text{'}Supervisior\text{'})$
$\wedge (F(checkCreditWorthinessNotSucceed)$
$\wedge (checkCreditWorthinessNotSucceed$
$\rightarrow X(CreateSuspenseFile))) \wedge (F\ (Create$
$SuspenseFileNotSucceed) \wedge (CreateSuspense$
$FileNotSucceed \rightarrow F(NotifyManager)))))$

In regard to $R6'$, *checkCreditWorthinessNotSucceed* represents the checking point of the success of the previous activity, i.e. *checkCreditWorthiness.Role('Supervisor')*. Similarly, *CreateSuspenseFileNotSucceed* is the checking point of the success of *CreateSuspenseFile* activity. The generated LTL formula captures the static semantics of compensable compliance rules and ensures that it is structurally modelled in the relevant business process model.

5.7 Support for non-monotonic requirements

In real-life scenarios, business processes are also subject to (non-monotonic) requirements that are less strict and can be overridden under certain pre-defined conditions. In these cases, a non-monotonic requirement is still considered satisfied if it is overridden by one of its pre-defined exceptions. We consider CRL's support to such requirements necessary to enable relaxations and thereby handling exceptional situations. With respect to the strictness of the condition, exceptions has two types [64]:

- A *strong exception* on the primary rule mandates that whenever the strong exception holds, the primary rule *must not* hold. For example, for a compliance requirement that demands 'checking of customer's bank privileges when a new loan request is received' can have a strong exception that *mandates* 'the check' *to be skipped* if the loan requester is a *trusted* customer (e.g. a customer to the bank for more than 10 years with a good history).
- A *weak exception* on the primary rule indicates that whenever the weak exception holds, the primary rule *may or may not* hold. For the same example, where the requirement demands 'checking of the customer's bank privileges when a new loan request is received' can have a weak exception that *allows* 'the check' to be performed as an *optional* task 'if the loan amount is less that 1 million Euro and the requester is a *trusted* customer'.

Following the approach in [64], in CRL, the exceptions are specified as separate rules and linked to the primary rule via *labels* (i.e. *label* and *exception label* constructs in Fig. 3). For example, compliance requirement *Req.7* in Table 2 is a non-monotonic requirement and can be represented in CRL as follows:

Req.7: *Checking banking privileges is optional for trusted (gold) customers, which is followed by checking the customer loan risk. If a trusted (gold) customer's loan request is less than 1M Euros, the evaluation of the loan risk must not be performed*

$\boldsymbol{R7}$: $([R_7^1]CheckCustomerBankPrivilege$
$LeadsTo\ [[R_7^2]]\ EvaluateLoanRisk)$
$\boldsymbol{R_7^1}$: $(Customer.Type = \text{'}TrustedGold\text{'})\ isUniversal$
$\boldsymbol{R_7^2}$: $(Customer.Type = \text{'}TrustedGold\text{'}\ And\ Loan.$
$Amount < \text{'}1M\text{'})\ isUniversal$

$R7$ captures the primary rule, and R_7^1 and R_7^2 represent weak and strong exceptions of the primary rule $R7$, respectively. (R_7^1 is put in a *single* square brackets indicating that it is a weak exception, while R_7^2 is enclosed between *double* square brackets indicating that it is a strong exception).

The transformation of a non-monotonic expression results in a single LTL formula that combines the primary rule with its exceptions. To simplify the presentation of the mapping, it is illustrated in two steps. First, the non-monotonic expression (the primary rule and its exception rules) is mapped to the LTL rules augmented by exception labels. For example, applying the mapping rules from Table 3 on the expression $R7$, the generated (augmented) LTL rules are as follows:

$\boldsymbol{R7}$: $G\ ([R_7^1]\ CheckCustomerBankPrivilege$
$\rightarrow\ [[R_7^2]]\ F\ (EvaluateLoanRisk))$
$\boldsymbol{R_7^1}$: $G(Customer.Type = \text{'}TrustedGold\text{'})$
$\boldsymbol{R_7^2}$: $G(Customer.Type = \text{'}TrustedGold\text{'}\ \Lambda\ Loan.$
$Amount < \text{'}1M\text{'})$

Second, (augmented) LTL rules are mapped into plain LTL formula following *Algorithm 1* (which extends [64]) given below based on the following definitions:

- Assume that L is the set of labels to be used to define corresponding strong and weak exceptions.
- Let R be the label of the primary rule.
- Primary and exception rules constitute a set of rules in the form of $< e : f >$, where e is the label, and f is the body of the rule.
- Let $< e_1 : f_1 >$ be an augmented LTL rule. If $e_2 \rightarrow L$ occurs in the body of $< e_1 : f_1 >$, then e_2 *depends* on e_1. The *dependency relation* is a transitive relation.
- An augmented LTL rule is *Loop Free* if and only if no label in L depends on itself. Based on these concepts, Algorithm 1 can be specified as:

Algorithm 1:
Mapping scheme of a non-monotonic expression into LTL
Input: L (set of rule labels), R (primary rule label)
Input: Set of augmented LTL rules T (the primary rule along with its exceptions)
Output: R (resulting LTL rule combining rules in T)

1. Let $< e : f_1 >, < e : f_2 >, \cdots, < e : f_n >$ be all the rules in T, where e is the label, such that $e \in [R] \cup L$. A formula $f_1 \bigvee f_2 \vee \cdots \vee f_n$ is constructed which is called $E(e)$. This is done for all labels if the set of rules with e is not empty
2. If label a_1 depends on a_2, and $E(a_1)$ is defined. All occurrences of $[a_1](f)$ in $E(a_2)$ will be replaced with $E(a_1) \vee f$. The revised formula is still called $E(a_2)$
3. If label a_1 depends on a_2, and $E(a_1)$ is defined. All occurrences of $[[a_1]](f)$ in $E(a_2)$ will be replaced with $E(a_1) \rightarrow \neg f$. The revised formula is still called $E(a_2)$
4. Steps 2 and 3 are repeated iteratively until no labels e depending on R, while $E(e)$ is not empty occurs in $E(R)$
5. Finally, in $E(R)$, all the remaining $[r](f)$ and $[[r]](f)$ (exceptional rules that are not defined) are replaced with f. The generated LTL formula is still called R

Applying *Algorithm 1* to $R7$ and its exceptions, step 1 (in *Algorithm 1*) is skipped since no exceptions have the same label in the exception rules set. By applying step 2 (in *Algorithm 1*) to replace the weak exception R_7^1 in $R7$, the resulting LTL formula is as follows:

$$\boldsymbol{R7'}: G((Customer.Type = \text{`}TrustedGold\text{'} \vee CheckCustomerBankPrivilege) \rightarrow [[R_7^2]] F(EvaluateLoanRisk))$$

Next, by applying step 3 to replace the strong exception R_7^2 in $R7'$, the resulting LTL formula is as follows:

$$\boldsymbol{R7'}: G((Customer.Type = \text{`}TrustedGold\text{'} \vee CheckCustomerBankPrivilege) \rightarrow ((Customer.Type = \text{`}TrustedGold\text{'} \wedge Loan.Amount\text{`}1M\text{'}) \rightarrow \neg F(EvaluateLoanRisk))$$

Steps 4 and 5 of the algorithm are skipped as $R7'$ does not contain more exceptions. The resulting $R7'$ is a pure LTL formula that can be fed into the subsequent design-time verification phase for compliance checking (Sect. 6).

6 Prototypical implementation

An implementation of the proposed design-time compliance management framework is a challenging yet necessary step to help validating the soundness of our approach. We have developed a comprehensive experimental tool suite[3] for design-time BP compliance management, implementing the approach described in the above sections. Figure 4 presents three main components of the tool suite and their relationships: *Compliance Rule Manager* (*CRM*), *Design-time Compliance Verification Manager* (*DCVM*), and *Web Service Analysis Tool* (*WSAT*).

6.1 Compliance Rule Manager (CRM)

The *Compliance Rule Manager* is a standalone application developed with Microsoft Visual Studio environment (with Visual Studio 2008 SDK) using C# programming language. The upper left-hand side part of Fig. 4 depicts the internal architecture of the Compliance Rule Manager, its components, and their interaction with the *business process* and *compliance repositories*. Although conceptually separated, these repositories reside in a shared environment, which is implemented using Oracle database (version.9i).

The CRM comprises two sub-components: *Compliance Rule Modeller* and *Text Template Transformation Toolkit*. The *Compliance Rule Modeller* is a graphical modeller that is used to visually design and create *pattern-based expressions* of compliance requirements in a drag-and-drop fashion. A

[3] *BPCM—Business Process Compliance Management Tool Suite:* http://eriss.uvt.nl/compas/.

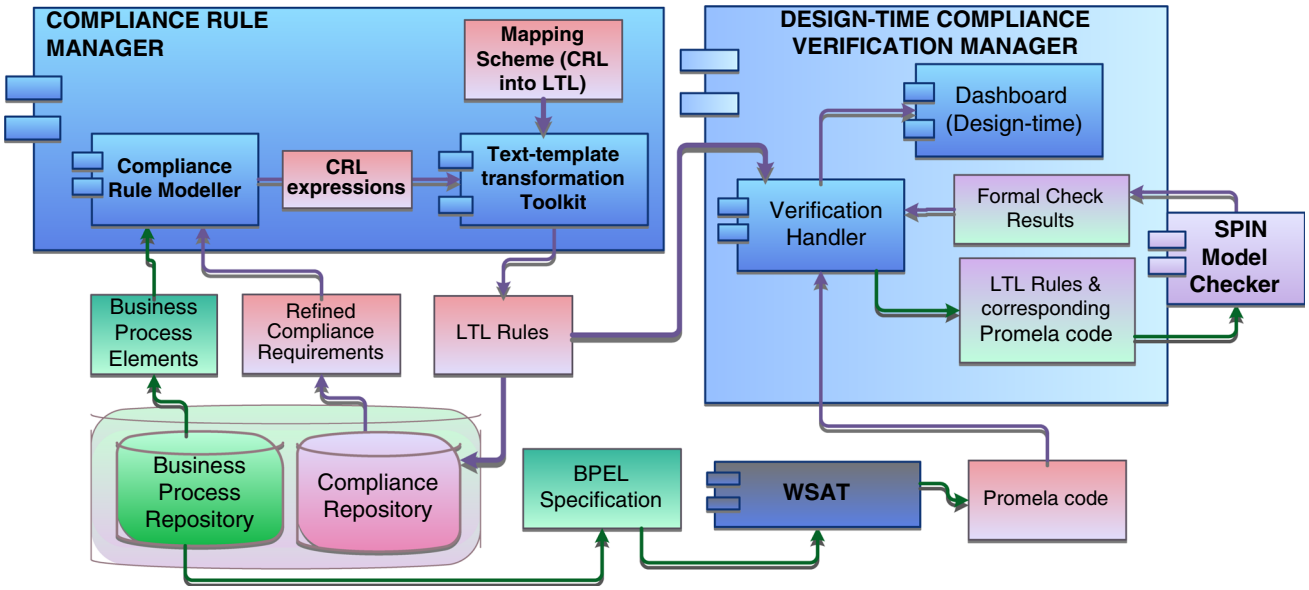

Fig. 4 A high-level architectural view of the interacting components of the tool suite

screenshot from the Compliance Rule Modeller is shown in Fig. 5. The patterns and the operand types are situated on the left side of the interface on the toolbox. The users drag and drop these constructs on the drawing canvas to build pattern-based expressions. The drawing canvas is divided into swimlanes, each belonging to a refined compliance requirement retrieved from the *Compliance Repository*. As described in Sect. 5.1, *patterns* and *operands* are the main elements that comprise pattern-based expressions. The operands are in the form of business process elements (such as activities, roles, data objects, events), their attributes, or conditions on them.

When an operand type, such as an activity, is selected from the toolbox and dragged onto the swimlane, the *Compliance Rule Modeller* retrieves the selected type of business process elements available in the *Business Process Repository* and presents the list to the user for selection (e.g. 'Select an Activity' dialogue box shown in Fig. 5).

Figure 5 shows the pattern-based representations of compliance requirements *Req.3* and *Req.7* of Table 2. Each requirement is represented with one or more pattern-based expressions that capture its semantics and each expression is enclosed in a sub-expression container. For example, the upper swimlane in Fig. 5 graphically represents requirements *Req.3* of the running example as described in Table 2 and as represented in expressions *R3.1 to R3.3* in Sect. 5.3 in textual form.

The *Text Template Transformation Toolkit* enables the automatic generation of formal compliance rules (as LTL/MTL formulas) based on the visual pattern-based expressions. This component is implemented using Microsoft Visual Studio 2008—T4. The output is an XML document that contains compliance rules as LTL/MTL formulas and their properties, as well as pattern-based expressions in text formats. The output file is parsed and resulting data is forwarded to the compliance repository together with the references to relevant compliance requirements.

Examples of formal compliance rules generated using the Text Template Transformation Toolkit is shown in Fig. 6. These sample LTL formulas are based on the pattern-based expressions given in Fig. 5 (originated from Req.3 and Req.7 in Table 2). As shown in Fig. 4, generated formal compliance rules are input to the subsequent compliance verification and monitoring phases of the business process compliance management framework.

6.2 Web Service Analysis Tool (WSAT)

WSAT [47] is an open source tool that implements the BPEL mapping approach proposed in [46]. Accordingly, a BPEL specification is first abstracted to a *Guarded Automaton* (*GA*) representing the global sequence of messages exchanged between participating services. *GA* is a finite state automaton (FSA) augmented with an unbounded queue for incoming messages. Guards can be specified on transitions that are represented as XPath expressions, which enable rich data manipulation and analysis.

Next, GA is mapped to *Promela* code. *Promela* (program meta language) is the input language accepted by SPIN model checker [8]. As advocated in [46], having BPEL specifications specified as GA as an intermediate step decouples the BP specification languages and formal verification tools from the translator. In addition, it enables the application of other static analysis techniques, e.g. synchronisability and realisability analysis (we refer the reader to [46] and [47] for further details). To be able to verify resource allocation and authorization constraints, *Roles* are captured from the '*Part-*

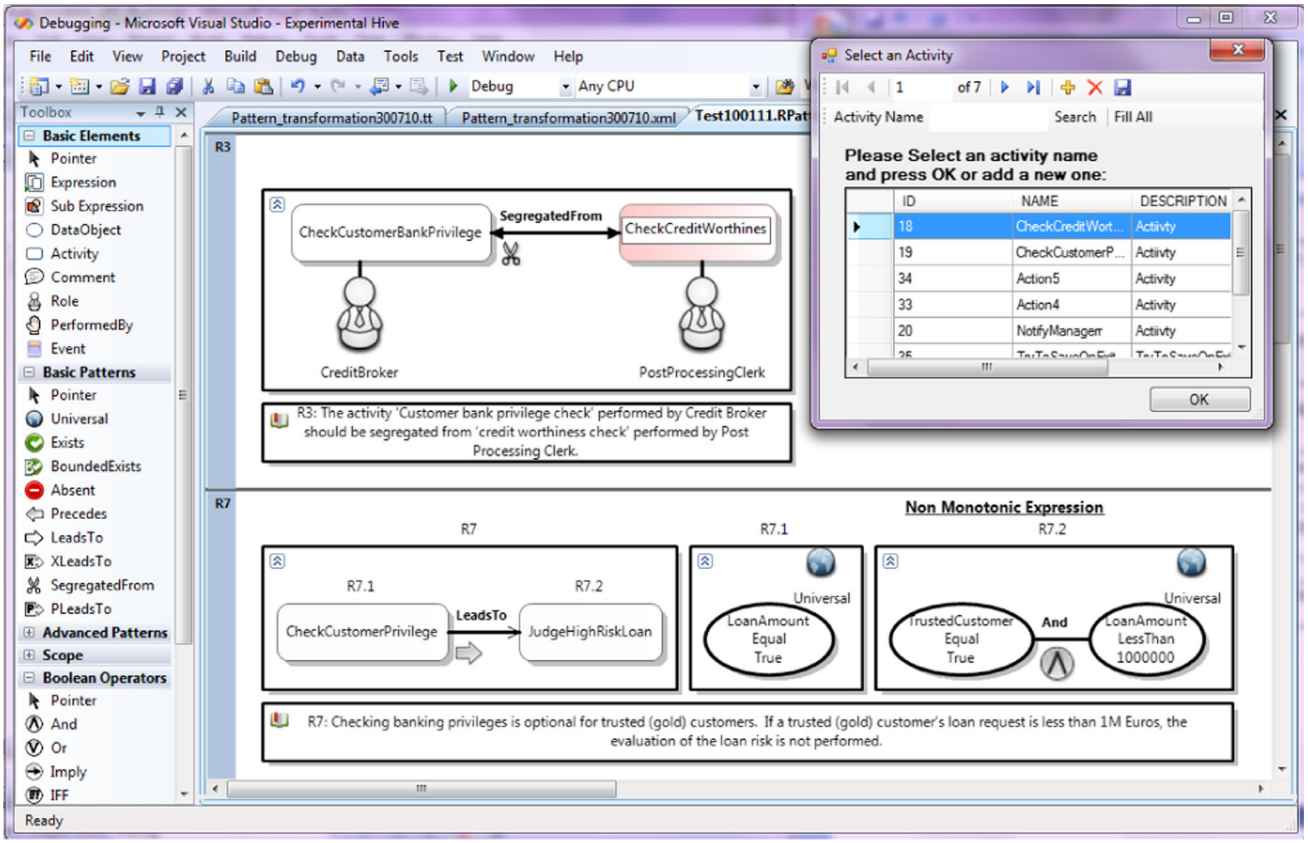

Fig. 5 A user interface from the compliance rule manager

nerRole' attribute of the '*partnerLink*' element in the BPEL specification. Partner links are used to link a BPEL specification to its interacting services. The corresponding partner link is then linked to BPEL basic activities (i.e. *Invoke*, *Receive*, *and Reply*) via the '*partnerLink*' attribute.

6.3 Design-Time Compliance Verification Manager (DCVM)

DCVM is a web-based environment (http://eriss.uvt.nl/compas), coded in 'PHP' (scripting language). It interacts with the *Compliance* and *BP Repositories*, which share the same environment running Oracle database (version.9i). As shown in Fig. 4, DCVM comprises two sub-components; firstly, the *Verification Handler* enables users to formulate compliance verification requests. It retrieves *formal compliance rules* (in LTL) from the compliance repository and BP specifications encoded in *Promela* code (WSAT output) and feeds those to SPIN for formal compliance checking. Secondly, the Verification Handler retrieves the checking results from SPIN and reports the possible underlying causes and caveats of detected violations based on the root-cause analysis approach we previously proposed in [51,52]. SPIN implements exhaustive state search as well as multiple optimized evaluation algorithms, e.g. partial-order reduction, hash-compact searches, which help to solve the typical state explosion problem.

Delivery of results in acceptable time is of utmost important for the proposed automated BP compliance verification approach to be adopted in practice. In order to investigate the performance of the tool suite, we conducted an experiment over the running scenario that we used in this paper. The experiment was conducted on a machine with an Intel Pentium 4 processor 1.7 GHz, and 4 GB RAM, with Microsoft Windows 7 operating system installed. The following results are reported: WSAT took around 49 s on the average to transform the BPEL process of the running scenario into its corresponding *Promela* code. The checking of the generated LTL rules capturing compliance requirements Req.1–7 presented in Table 2 consumed between 0.035 and 15 s for each of the rules. It is worth noting that 15 s were only consumed in the checking of the compensation compliance requirement R6. The average time of all other rules is 4.6 s.

The peak memory use was 1,026 MB. We believe that the results we received for the scenario are acceptable in terms of time performance. However, considering the complexity of real-life cases with large models and more complex compliance rules, the need for more efficient verification and analysis tools running on more powerful platforms is evident.

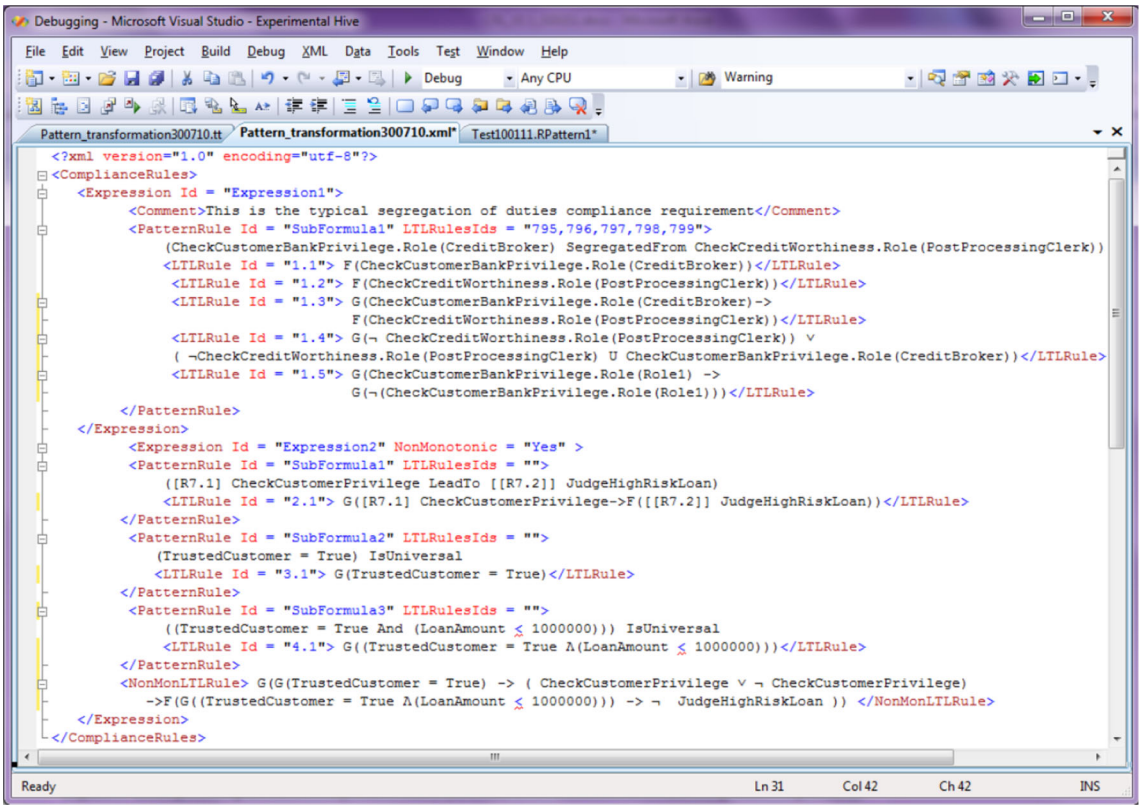

Fig. 6 Examples of generated compliance rules (in LTL) to be transferred to the compliance repository

With regard to the satisfaction of each of the LTL rules, SPIN indicated the following verification results:

- Compliance requirement Req.1 is violated.
- Compliance requirement Req.2 is satisfied.
- Compliance requirement Req.3 is satisfied.
  - Compliance rule R3.1 is satisfied.
  - Compliance rule R3.2 is satisfied.
  - Compliance rule R3.3 is satisfied.
  - Compliance rule R3.4: could not be checked (details are given below)
- Compliance requirement Req.4 is satisfied.
  - Compliance rule R4.1 is satisfied.
  - Compliance rule R4.2 is satisfied.
  - Compliance rule R4.3 is satisfied.
  - Compliance rule R4.4 is satisfied.
  - Compliance rule R4.5 is satisfied.
- Compliance requirement Req.5:
  - Compliance rule R5.1 is satisfied.
  - Compliance rule R5.2: could not be checked (details are given below)
- Compliance requirement Req.6 is violated.
- Compliance requirement Req.7 is violated.

It was not possible to check the compliance rule R5.2, since the exact time where each BP activity takes place is not modelled in BPEL, which is considered as a limitation. Timeouts, however, can be modelled in BPEL, which enabled us to verify such rules (e.g. R5.1). An extension to BPEL is required to capture these time annotations that capacitate the checking of these time-dependent compliance constraints during design time, which is considered as an open research point. Nevertheless, compliance rule R5.2 and similar time-dependent compliance constraints can be specified using CRL timed patterns and can be reserved for the subsequent run-time monitoring, where time information is available. Analogously, compliance rule R3.4 could not be checked due to the absence of user's information and the same rule can be reused for run-time verification.

7 Evaluation

The utility of a design artefact must be rigorously demonstrated via well-executed evaluation methods [65]. Observational methods, such as case studies and field studies, allow an in-depth analysis of the artefact and the monitoring of its use in multiple projects within the technical infrastructure of the business environment.

In Sect. 4, we have introduced two case studies that involved processes operating in e-business and banking domains. The case studies involved the specification of the compliance requirements (using the CRL) that are applied on companies' processes with the main objective of investigating the *applicability* and *expressiveness* of compliance patterns that have been introduced as an integral part of the compliance management approach and implemented in the *Compliance Rule Manager* Software tool. The case studies also allowed us to experiment with the use of the Compliance Rule Manager for building graphical representations of pattern-based expressions and automatically generating corresponding formal compliance rules.

For each case study, the case study team—comprising both compliance and business experts—followed a three-phased approach, which involves the following activities:

- In the first phase, the team analysed the normative specifications (i.e. compliance sources, such as regulations, legislations, domain standards, internal policies) that the companies in the case studies were to comply with and identified *high-level compliance requirements* imposed on the processes. This phase resulted in 52 high-level requirements for the first case and 7 requirements (some of which are listed in Table 2 as examples) in the second case study.
- In the second phase, grounded on the high-level requirements that were identified and the current design of the processes, the team iteratively refined company-specific interpretation of these requirements in textual forms (examples of these refined/internalized compliance requirements are also given in Table 2). The requirements are also categorized with respect to the control approach (preventive or detective) and control means (process, technical, or physical). The team generated 122 refined requirements in the first and 13 requirements in the second case study.
- The final phase involved the construction of pattern-based expressions (of *process* requirements that are applicable to *preventive* approach) through the use of the *Compliance Rule Manager* Software tool.

Table 7 gives the type and number of compliance requirements covered within the case studies and whether the case study participants were able to express these requirement effectively using patterns introduced in this paper.

As shown in Table 7, the requirements that are classified as *process* and can be handled using *preventive* control approaches constituted the majority. These requirements were of particular interest to us, as this type of constraints is the main target for the pattern-based representations and formalization. They involved rules concerning mainly segregation of duties, access rights, condition-based sequencing of activities, and data processing requirements. Under this category of requirements (i.e. preventive process), the case study teams were successful in expressing all *segregation-of-duties* requirements in both case studies. Requirements concerning *access rights / authorizations, activity sequencing* were also successfully expressed using the CRL, as it has rich constructs to capture the constraints on the control flow and resource (e.g. role/user assignments) aspects of processes.

Participants were able to express 72 of such requirements (out of 82) using the Compliance Rule Modeller tool, while 10 requirements that were tagged as 'preventive process' could *not* be expressed using available patterns. These requirements mainly involved constraints regarding *data processing*, e.g. rules that are related to the structure and integrity of the data manipulated within the processes. They typically demanded for sequential numbering of certain business objects, such as orders or invoices.

Despite the limitations discussed above, we can conclude that within the preventive-process category of compliance requirements, the proposed patterns were effective means for expressing compliance requirements of certain concerns. In particular, the concerns that influence the control flow, resource (roles, actors, etc.) and temporal aspects of BP are accurately expressed by patterns.

The *process* requirements that are controlled following the *detective* approaches involved mainly *management reviews*,*reconciliations* and*performance monitoring*. The support by the CRL patterns for the representation and formalization of such requirements is inherently missing, as their verification often requires manual checks to guarantee assurance. *Technical* requirements were *preventive* and involved the use of specific techniques for data encryption, data retention and authentication mechanisms. *Physical* requirements were also *preventive* in nature that demanded locks or guards to protect against unauthorized access to physical assets. CRL is not designed to address these types of requirements.

In addition, we can compare our approach to major related-work efforts that are discussed in Sect. 2 (and is drawn as a comparison in Table 1). Accordingly, we can conclude that our approach is the only one that satisfies all the criteria considered for the comparison. More specifically:

- The approach presented in this paper incorporates a high-level graphical pattern-based compliance specification language that addresses the usability concern of formal languages.
- The approach is based on a rigour formal foundation of temporal logic (i.e. LTL), which is widely experimented on large-scale systems in various application domains,

Table 7 Categories and numbers of compliance requirements covered in the case studies

Type of controls	Number of comp. requirements			Effectively expressed using *CRL patterns*?	
	Case St. 1: internet reseller	Case St. 2: loan processing	Total	Yes	No
Process					
Preventive[a]	71	11	82	72	10
Segregation of duties (SOD)[a]	30	2	32	32	–
Access rights/authorizations	10	3	13	13	–
Activity sequencing	21	6	27	27	–
Data processing	10	–	10	–	10
Detective	38	–	38	–	38
Management reviews and reconciliations	30	–	30	–	30
QoS/performance monitoring	8	–	8	–	8
Technical					
Preventive	5	2	7	–	7
Detective	–	–	–	–	–
Physical					
Preventive	8	–	8	–	8
Detective	–	–	–	–	–
Total	122	13	135	72	63

[a] Segregation of duties (SOD) involves the separation of the control of a process among different roles/individuals in order to decrease the risk of fraud/mistake by one person

and incorporates sophisticated verification and analysis tools, such as model checkers.

- The approach considers the Business Process Execution Language (BPEL) standard, which is the de fact standard for web services orchestration.
- The approach supports the specification and verification of three out of four classes of compliance requirements, i.e. control flow, employed resources and time constraints. Support of data constraints is an ongoing work.
- The approach provides efficient solutions to support non-monotonic requirements and compensations, and to reason about root causes of detected compliance violations and guide the user in their resolution.

Furthermore, as part of the EU COMPAS project, we have received positive feedback from COMPAS industrial partners (PricewaterhouseCoopers Netherlands and Thales Services France) with whom we worked closely to validate and evaluate the proposed solutions. The feedback we received was based on their continual meetings with their customers, mainly from the banking domain. Carrying out surveys to investigate the perceived usefulness of the approach is considered one of the future work directions which we highlighted in Sect. 9.

8 Conclusions and lessons learnt

Business processes form the foundation for all organizations, and as such, are impacted by industry regulations. Without explicit BP definitions, effective and expressive compliance frameworks, organizations face litigation risks and even criminal penalties. Compliance management should be one of the integral parts of BPM, such that compliance requirements should be based on a formal foundation of a logical language to enable future reasoning techniques for verifying and ensuring BP compliance starting from the early stages of the business process design.

This paper contributes by a comprehensive design-time compliance management framework that encompasses in its core a high-level and rich pattern-based compliance specification language for facilitating the formal specification of compliance requirements on BPs. The case studies we conducted showed the expressive power of the patterns that we proposed. We believe that using graphical patterns could significantly facilitate the work of the compliance and BP experts by shielding the complexity of the formalizms off from the end-users.

CRL is formally based on temporal logic, i.e. LTL/MTL. We consider CRL as an extensible open language as mapping rules from CRL pattern classes into other formal languages may also be defined. This will enable the specification of some requirements that are not expressible in LTL. Since the

expressive power of CRL is limited to the expressive power of LTL, then these requirements are also inexpressible in CRL.

As a proof-of-concept, an integrated tool suite has been developed, where CRL is implemented as a graphical prototype, which enables the specification of compliance requirements intuitively using patterns in a drag-and-drop fashion and automates the process of transforming these definitions into logical formulas. The approach is further evaluated and validated by its application on two case studies performed with two industrial companies.

Managing compliance of business processes requires a multi-faceted approach as the problem involves not only technical aspects rooted in various fields that have to be bridged (such as computer science, business process management, formal methods, and legal studies) but also social and organizational aspects as it highly involves knowledge work. No matter how sophisticated the offered solutions and the underlying technologies are, BP compliance management cannot be fully automated. Having efficient techniques and solutions in place can only facilitate and improve the quality of the work involved.

As also shown by the case studies we conducted, the automated verification and monitoring of compliance is possible only for a certain segment of requirements. Several technical and process- related requirements necessitate checks and controls that have to be performed manually by compliance experts.

Automated tools and techniques should also allow for flexible solutions and human intervention to cope with the non-monotonicity of real life with frequent exceptions. Besides, wide adoption by the industry is only possible if the automated techniques and solutions allow compliance to be managed (cost) effectively. In addition, such solutions should not only provide plain verification results but also give insight into the root causes of compliance violations and propose alternative solutions as remedies.

9 Future work

Future research and development work is ongoing in several directions to enhance and fully support the comprehensive compliance management framework. Ongoing work focuses on the integration of the design-time compliance management approach proposed in this article with the subsequent run-time monitoring and management. In particular, the work presented in this paper will be integrated with the compliance monitoring approach introduced in [66] on the basis of the Compliance Request Language. That is, from pattern-based CRL expressions, design-time and run-time formal rules can be automatically generated for both design-time and run-time compliance assurances.

Ongoing work also involves basing the compliance management framework on semantic repositories. This involves building a set of inter-related semantic ontologies (e.g. business process ontology, compliance ontology) using the Ontology Web Language (OWL2.0) standard as part of a central compliance management knowledge base. This will allow us to conduct a set of preliminary structural analysis using the reasoning tools associated with these technologies and ensures the ontological alignment between compliance and business specifications.

To facilitate the communication between different stakeholders, we are working on the interpretation and encoding of the anti-money laundering regulation using the Semantics of Business Vocabulary and Business Rules (SBVR) standard [67]. Future work will include the integration of the resultant SBVR specifications [68] with the compliance management approach presented in this paper to facilitate the compliance refinement process we previously proposed in [48] and [49].

On a more fine-grained perspective, our analysis of LTL with respect to a set of desired features of CRL [59] revealed a number of limitations. In particular, a support for checking the *consistency* among compliance requirements is required, as conflicts and contradictions may rise between compliance rules, especially when they originate from various compliance sources.

The case studies we conducted investigated the applicability and expressiveness of the compliance patterns that we proposed in this paper. Our future work should also consider experiments to explore their perceived usefulness and efficiency, possibly through application of some prominent technology adoption models. The validation of the proposed approach will be further intensified by its application on various case studies on prospective users of the patterns and developed tool suite.

Acknowledgments The authors gratefully acknowledge PricewaterhouseCoopers (Netherlands), Thales Services (France), and other COMPAS project partners for their effort in providing and participating in the case studies and scenarios, and their valuable contributions. Special thanks to Dr. Guido Governatori (NICTA, Australia) for reviewing the paper and for his valuable comments.

References

1. SOX: Sarbanes-Oxley Act of 2002. In: Congress, U.S. (ed.), (2002)
2. Bank for International Settlements: Basel III: International framework for liquidity risk measurement, standards and monitoring (2010)
3. Accutiy. Visualising trends in anti-money laundering compliance. http://www.accuity.com/industry-updates/free-resources/trends-in-aml-compliance-infographic/. Accessed 28 Nov 2013
4. Ernst & Young: The Top 10 Risks For Business. The Ernst & Young Business Risk Report (2010)
5. Hartman, T.: The Cost of Being Public in the ERA of Sarbanes-Oxley. Foley & Lardner LLP (2006)

6. Goedertier, S., Vanthienen, J.: Designing compliant business processes with obligations and permissions. In: International Business Process Management Workshops (BPM), Austria, pp. 5–14 (2006)
7. Sadiq, S., Governatori, G., Naimiri, K.: Modeling control objectives for business process compliance. In: Business Process Management-BPM'09 Proceedings, pp. 149–164 (2007)
8. Holzmann, G.: The model checker SPIN. IEEE Trans. Softw. Eng. **23**, 279–295 (1997)
9. Ly, L.T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems. Inf. Syst. Front. **14**(2), 195–219 (2012)
10. Halle, S., Villemaire, R., Cherkaoui, O.: Specifying and validating data-aware temporal web service properties. IEEE Trans. Softw. Eng. **35**, 669–683 (2009)
11. Giblin, C., Liu, A., Muller, S., Pfitzmann, B., Zhou, X.: Regulations expressed as logical models. In: 18th International Annual Conference of Legal Knowledge and Information Systems, Belgium, pp. 37–48 (2005)
12. Eshuis, R.: Symbolic model checking of UML activity diagrams. ACM Trans. Softw. Eng. Methodol. **15**, 1–38 (2006)
13. Wang, H.J., Leon Zhao, J.: Constraint-centric workflow change analytics. Decis. Support Syst. **51**, 562–575 (2011)
14. Abouzaid, F., Mullins, J.: A calculus for generation, verification, and refinement of BPEL specifications. Electron. Notes Theor. Comput. Sci. (ENTCS) **200**, 43–65 (2008)
15. Awad, A., Gore, R., Thomson, J., Weidlich, M.: An iterative approach for business process template synthesis from compliance rules. In: 23rd International Conference on Advanced Information Systems, Engineering, pp. 406–421 (2011)
16. Yu, J., Han, Y., Han, J., Jin, Y., Falcarin, P., Morisio, M.: Synthesizing service composition models on the basis of temporal business rules. J. Comput. Sci. Technol. **23**, 885–894 (2008)
17. Liu, Y., Muller, S., Xu, K.: A static compliance-checking framework for business process models. IBM Syst. J. **46**, 335–361 (2007)
18. Awad, A., Weidlich, M., Weske, M.: Specification, verification and explanation of violation for data aware compliance rules. In: 7th International Conference on Service Oriented Computing (ICSOC-Service Wave'09), vol. 5900, pp. 500–515. Springer, Berlin (2009)
19. Geist, D.: The PSL/sugar specification language: a language for all seasons. In: The Correct Hardware Design and Verification Methods Conference, pp. 21–24 (2003)
20. Khaluf, L., Gerth, C., Engels, G.: Pattern-based modeling and formalizing of business process quality constraints. In: CAiSE'11, pp. 521–535 (2011)
21. Yu, J., Manh, T., Han, J., Jin, Y.: Pattern based property specification and verification for service composition. In: K.A. et al. (eds) WISE 2006, LNCS-4255, pp. 156–168. Springer, Berlin (2006)
22. Dwyer, M., Avrunin, G., Corbett, J.: Property specification patterns for finite-state verification. In: 2nd International Workshop on Formal Methods on Software, Practice, pp. 7–15 (1998)
23. Pelliccione, P., Inverardi, P., Muccini, H.: CHARMY: a framework for designing and verifying architectural specifications. IEEE Trans. Softw. Eng. **35**, 325–346 (2009)
24. Ramezani, E., Fahland, D., van der Aalst, W.: Where did i misbehave? Diagnostic information in compliance checking. In: 10th International Conference on Business Process Management (BPM), pp. 262–278. Springer, Berlin (2012)
25. Accorsi, R., Sato, Y.: Automated certification for compliant cloud-based business processes. Bus. Inf. Syst. Eng. (BISE) **3**, 145–154 (2011)
26. Accorsi, R., Lehmann, A.: Automatic information flow analysis of business process models. In: 10th International Conference on Business Process Management (BPM), pp. 172–187. Springer, Berlin (2012)
27. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: EDOC'07, pp. 287–300 (2007)
28. Pesic, M., van der Aalst, W.: A declarative approach for flexible business processes management. In: BPM'06 Workshops (2006)
29. Konrad, S., Cheng, B.: Real-time specification patterns. In: International Conference on Software Engineering (ICSE'05), USA, pp. 15–21 (2005)
30. Giblin, C., Muller, S., Pfitzmann, B.: From Regulatory Policies to Event Monitoring Rules. Zurich Research Laboratory, Zurich (2006)
31. Gruhn, V., Laue, R.: Specification patterns for time-related properties. In: 12th Int'l Symposium on Temporal Representation and Reasoning, pp. 198–191 (2005)
32. Wolter, C., Schaad, A.: Modeling of task-based authorization constraints in BPMN. In: Business Process Management (BPM 2007), pp. 64–79. Springer, Berlin (2007)
33. Ahn, G., Sandhu, R., Kang, M., Park., J.: Injecting RBAC to secure a web-based workflow system. In: RBAC '00, pp. 1–10 (2000)
34. Governatori, G., Milosevic, Z., Sadiq, S.: Compliance checking between business processes and business contracts. In: 10th International Enterprise Distributed Object Computing Conference (EDOC 2006), pp. 221–232 (2006)
35. Governatori, G., Rotolo, A.: Justice delayed is justice denied: logics for a temporal account of reparations and legal compliance. In: Computational Logic in Multi-Agent Systems, vol. 6814, pp. 364–382 (2011)
36. Thomas, F.: Constructing legal arguments with rules in the legal knowledge interchange format (LKIF). In: Computable Models of the Law, Languages, Dialogues, Games, Ontologies, vol. 4884, pp. 162–184 (2008)
37. Palmirani, M., Governatori, G., Contissa, G.: Modelling temporal legal rules. In: International Conference on Artificial Intelligence and Law, pp. 131–135 (2011)
38. Governatori, G., Olivieri, F., Scannapieco, S., Cristani, M.: Designing for compliance: norms and goals. In: 5th International Conference on Rule-Based Modeling and Computing on the Semantic Web, pp. 282–297 (2011)
39. Governatori, G., Rotolo, A.: Bio logical agents: norms, beliefs, intentions in defeasible logic. J. Auton. Agents Multi Agent Syst. **17**, 36–69 (2008)
40. Markovic, I., Pereira, A.C., Stojanovic, N.: A framework for querying in business process modelling. International Multikonferenz Wirtschaftsinformatik, Germany, pp. 1703–1714 (2008)
41. Beeri, C., Eyal, A., Kamenkovich., S.: Querying business processes. In: 32nd International VLDB Conference, Korea, pp. 343–354 (2006)
42. Kühne, S., Kern, H., Gruhn, V., Laue, R.: Business process modeling with continuous validation. J. Softw. Evol. Process **22**, 547–566 (2010)
43. Delfmann, P., Herwig, S., Lis, L., Stein, A., Tent, K., Becker, J.: Pattern specification and matching in conceptual models: a generic approach based on set operations. Enterp. Modell. Inf. Syst. Arch. **5**, 24–43 (2010)
44. Awad, A.: BPMN-Q: A language to query business processes. In: 2nd International Workshop on Enterprise Modelling and Information Systems Architectures: Concepts and Applications (EMISA), Germany, pp. 115–128 (2007)
45. Elgammal, A., Turetken, O., van den Heuvel, W., Papazoglou, M.: Towards a comprehensive design-time compliance management: a roadmap. In: 15 International Business Information Management Conference (15th IBIMA), Egypt, pp. 1480–1484 (2010)
46. Fu, X., Bultan, T., Su, J.: Analysis of Interacting BPEL Web Services. World Wide Web (WWW), pp. 621–630. ACM Press, USA (2004)

47. Fu, X., Bultan, T., Su, J.: WSAT: a tool for formal analysis of web services. In: 16th International Conference on Computer Aided Verification, USA, pp. 510–514 (2004)
48. Turetken, O., Elgammal, A., van den Heuvel, W.J., Papazoglou, M.: Enforcing compliance on business processes through the use of patterns. In: 19th European Conference on Information Systems (ECIS 2011), Finland (2011)
49. Turetken, O., Elgammal, A., van den Heuvel, W., Papazoglou, M.: Capturing compliance requirements: a pattern-based approach. IEEE Softw. **29**, 28–36 (2012)
50. COSO: Internal Control: Integrated Framework. The Committee of Sponsoring Organizations of the Treadway Commission (1994)
51. Elgammal, A., Turetken, O., van den Heuvel, W., Papazoglou, M.: Root-cause analysis of design-time compliance violations on the basis of property patterns. In: 8th International Conference on Service-Oriented Computing (ICSOC'10), USA, pp. 17–31 (2010)
52. Elgammal, A., Turetken, O., van den Heuvel, W.: Using patterns for the analysis and resolution of compliance violations. Int. J. Coop. Inf. Syst. **21**, 31–54 (2012)
53. COMPAS Project, Deliverable 2.1: State-of-the-Art in the Field of Compliance Languages (2008)
54. IFRS: International Financial Reporting Standards. International Accounting Standards Board (2001)
55. FINRA: The Financial Industry Regulatory Authority, "FINRA Manual" (2008)
56. COBIT: Control Objectives for Information and related Technology: COBIT, 4.1. IT Governance Institute (2007)
57. OCEG: GRC Capability Model, Ver 2.0. Open Compliance and Ethics Group (2009)
58. Elgammal, A., Turetken, O., van den Heuvel, W., Papazoglou, M.: On the formal specification of regulatory compliance: a comparative analysis. In: International Performance Assessment and Auditing in Service Computing Workshop, ICSOC'10 workshops, USA (2010)
59. Elgammal, A., Turetken, O., van den Heuvel, W., Papazoglou, M.: On the formal specification of business contracts and regulatory compliance. In: 4th Workshop on Formal Languages and Analysis of Contract-Oriented Software, EPTCS, Pisa, Italy. pp. 33–36 (2010)
60. Elgammal, A.: Towards a comprehensive framework for business process compliance. Ph.D. Dissertation. Information Management Department, Tilburg University, Tilburg University Press, pp. 284 (April 2012)
61. Pnueli, A.: The temporal logic of programs. In: 18th IEEE Symposium on Foundations of Computer, Science, pp. 46–57 (1977)
62. Armoni, R., Fix, L., Flaisher, A., Gerth, R., Ginsburg, B., Kanza, T., Landver, A., Mador-Haim, S., Singerman, E., Tiemeyer, A., Vardi, M., Zbar, Y.: The ForSpec temporal logic: a new temporal property-specification language. Lecture Notes In Computer Science, vol. 2280 (2002)
63. Alur, R., Henzinger, T.: Real-time logics: complexity and expressiveness. Inf. Comput. **104**, 35–77 (1993)
64. Baral, C., Zhoa, J.: Non-monotonic temporal logics for goal specifications. In: 20th International Intelligence Conference on Artificial Intelligence (IJCAI-07), India, pp. 236–242 (2007)
65. Hevner, A., March, S., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**, 75–105 (2004)
66. Sebahi, S.: Business process compliance monitoring: a view based approach. Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS), Ph.D. University Lyon 1, Lyon (2012)
67. OMG: Semantics Of Business Vocabulary And Business Rules (SBVR), Version 1.0. (2008)
68. Abi-Lahoud, E., Butler, T., Chapin, D., Hall, J.: Interpreting regulations in SBVR. In: RuleML (2013)

Amal Elgammal is a postdoctoral Researcher in the Governance, Risk Management and Compliance Technology Centre (GRCTC) at University College Cork. Her main research interests include Business process management, business process compliance management, service-oriented computing, business process monitoring and auditing, and ontology engineering. Contact her at aelgammal@ucc.ie.

Oktay Turetken is an Assistant Professor in the School of Industrial Engineering at Eindhoven University of Technology. He holds a Ph.D. in Information Systems. His research focuses on the modeling, improvement, maturity, governance and compliance of business processes. Contact him at o.turetken@tue.nl.

Willem-Jan van den Heuvel is a full professor in computer science at the Department of Information Systems, Tilburg University and managing director of the ERISS. His main research interests revolve around (cloud) service engineering, service governance, performance analytics of software-enabled service networks, and business transaction management. Contact him at W.J.A.M.vdnHeuvel@uvt.nl.

Mike Papazoglou is the chair of the Computer Science Department at Tilburg University. He's also the scientific director of the ERISS and the EC's Network of Excellence, S-Cube. His research interests include service-oriented computing, Web services, large-scale data sharing, business process management, and federated information systems and distributed computing.

Papazoglou has a PhD in microcomputers systems engineering from the University of Dundee, Scotland. He's a Golden Core Member and a Distinguished Visitor of the IEEE Computer Society. Contact him at m.p.papazoglou@uvt.nl.