A Mashup-based Framework for Business Process Compliance Checking

Cristina Cabanillas, Manuel Resinas and Antonio Ruiz-Cortés

Abstract—Business process compliance ensures that the business processes of an organisation are designed and executed according to the rules that enforce the compliance controls that govern the company. We faced the challenge of building a Business Process Compliance Management System (BPCMS) for a process-aware organisation that had to provide support for several needs that, despite having been identified in the literature, were only partially satisfied by existing approaches. The variability in the types of rules and their interpretation generally restricts the existing support for compliance checking to specific types of rules (e.g., rules affecting the control flow of the process), a specific phase of the business process management (BPM) lifecycle (e.g., design time or run time), or certain information systems (ISs) for data retrieval (e.g., process event logs). Motivated by this, we designed a conceptual framework for design-time and run-time compliance checking that relies on the use of *mashups* for rule specification and checking. It presents the following advantages: (i) an open-ended set of types rules can be specified by designing and connecting mashup components; (ii) (parts of) the definitions of the rules can be reused as needed; and (iii) the mashup-based compliance checking (MCC) system can be integrated with ISs of the organisation, enabling the verification of actual facts on actions performed during the execution of a process (e.g., the existence of a specific document in a concrete location). Design-time and run-time implementations of the framework were conducted and tested in a real setting.

Index Terms—Business process compliance, business process management, compliance checking, compliance framework, mashups

1 Introduction

Business process compliance ensures that the business processes of an organisation are designed and executed according to the regulations and internal policies that govern the company. Based on these regulations and internal policies a set of *compliance controls*[1], which must be observed by the organisation, can be defined. An example of compliance control is "On a quarterly basis, a tax completion checklist is used and signed by the Tax Manager and the Financial Controller" (*CC1*). Each compliance control can be further decomposed into a set of *compliance rules* or *constraints*[2] [1], [2]. These rules are specific checks that must be carried out in order to evaluate whether the control is being fulfilled. Therefore, rules specify how controls can be materialised in the context of the business processes and the Information Systems (ISs) of an organisation. Compliance rules relate to different aspects of a process (a.k.a. *business process perspectives*), such as the execution order of the activities (control flow or behavioural perspective); the data accessed and produced (informational perspective); or the people (human resources or just resources) that participate in the process (organisational perspective). For instance, *CC1* could be materialised in a rule like "Every four months, it is necessary to check whether a new document has been uploaded to the Document Management System (DMS) that contains the checklist completed and signed by the Tax Manager and the Financial Controller" (*CR1*).

Compliance rules can be checked at different phases of the Business Process Management (BPM) lifecycle [3], which results in two big modalities [4]. On the one hand, *forward* or *pre-mortem* compliance checking is the most proactive way to check compliance and comprises design time and run time. Design-time compliance checking is usually performed after process modelling to ensure that the process is compliant with the given rules before its execution, saving time and effort to business analysts. Run-time compliance checking (a.k.a. compliance monitoring or online auditing [5]) assesses rules at run time using data stored in the ISs of the organisation, so if a rule is violated or some problematic situation arises while running the process we might be able to solve the problem in time to avoid ending in a non-compliant state. On the other hand, *backward* or *post-mortem* compliance checking (a.k.a. offline auditing [5]) determines whether past instances of a process were compliant with rules from data stored in the ISs of the organisation. The result helps stakeholders to be prepared for potential future audits. The complexity of checking business process compliance has been acknowledged in the literature [6], [7].

In this paper we introduce a conceptual framework for forward compliance checking that, besides supporting most of the Compliance Monitoring Funcionalities (CMFs) identified and addressed in the literature [5], tackles the needs encountered when building and deploying a fully-fledged Business Process Compliance Management System (BPCMS) [8] in a real organisation. These needs, related to the heterogeneity of the ISs and the number and variability of compliance rules, can be covered if three requirements are met. The fact that the requirements have also been identified and partially addressed in the literature as we discuss next shows that their interest goes beyond a particular case.

- *Cristina Cabanillas was with the Institute for Information Business, Vienna University of Economics and Business, 1020 Vienna, Austria.*
- *Cristina Cabanillas, Manuel Resinas and Antonio Ruiz-Cortés are currently with the University of Seville, 41012 Seville, Spain. E-mails: cristinacabanillas@us.es, resinas@us.es, aruiz@us.es*

Manuscript received in August 2019; revised in –, 2020.

1. By *compliance* we always refer to *business process compliance*.
2. We will use *rule*, *compliance rule* and *constraint* interchangeably.

Requirement 1 (Req 1). Seamless integration with enterprise ISs. Several authors have already identified the need for compliance systems to obtain and integrate the required data from heterogeneous data sources [7], [9], [10]. Based on this observation, Giblin et al. identify the challenge of managing compliance rules in such distributed and heterogeneous IT environments [10], and Sadiq and Governatori highlight that integration technologies can be helpful in that regard [7]. This heterogeneity of sources is particularly relevant because approaches that assume homogeneous data sources may not be well suited for these scenarios. For instance, many proposals [11] consider as a unique data source the event logs of the process which, in turn, must be very complete in order to enable doing checks on the different process perspectives. However, in some cases, a direct evaluation of a system or database state may be more useful. This holds in cases in which it is inefficient to transform all data necessary to check the compliance rules into events [10] either because of the complexity of the transformation or because it might not be reasonable to include some required information in the log (e.g., when the check involves querying the content of a document).

Requirement 2 (Req 2). Support for an open-ended set of types of rules at design time and at run time. A compliance system must be able to specify and check rules regardless of the phase of the BPM lifecycle in which compliance is checked [12] and the business process perspectives involved. The challenge is that each business process perspective includes many types of rules [5]. This variability [4] leads to consider that supporting several formalisms for expressing rules is a desirable feature for frameworks. Moreover, even for the same type of rule, the system must provide the ability to fine tune its meaning. For instance, a well-known type of compliance rule is the separation of duties. While it is typically implemented to prevent the same person from performing two specific tasks of a process, under certain circumstances it is extended to enforce that the two people involved must also have different roles [5], [13]. Therefore, it is convenient to have a system that is not closed as to the types of rules that can be defined in it and the formalism used to check them.

Requirement 3 (Req 3). Rule specification reuse. The importance of compliance rule specification reuse has already been identified in several proposals [1], [12], [14]. All of them provide evidence on the existence of a high similarity between some compliance rules and the convenience of having a mechanism that enables reusing their specification instead of having to redefine them from scratch. This would help to save effort since only the adaptation to the business process at hand would be necessary [10], [14].

Research proposals have mostly focused on developing *specific techniques* for compliance checking, tending to disregard (some of) these 3 requirements. Moreover, assumptions usually made in terms of the required input data or the formalism used limit their ability to be extended to consider them. Exceptions to this are *frameworks* like [2], [15], [16], [17], [18] that focus on the problem of implementing a general architecture or infrastructure for compliance checking. However, as we discuss in Section 2, in the best case, they only partially address some of these requirements, which may limit their applicability in scenarios with high IS heterogeneity and a large variety of compliance rules.

Our proposal is related to such existing frameworks as we do not develop a new specific technique for compliance checking. We propose to use *mashups* [19] as a means to provide a homogeneous framework to specify rules and check design-time and run-time business process compliance, and we present an implementation of the framework that has been applied in a real setting within an R&D project. Mashups are easy to understand and use [20], and they can be implemented in many ways (e.g., by using spreadsheets [21]). Different formalisms can be combined in a single mashup provided that there is a way to connect the information resulting from a technique with the input required by another approach[3]. That enables their use to support an open-ended set of types of rules by integrating techniques of different nature (Req 2). Mashups offer flexibility, portability and reusability [19] so (part of) already defined rules can be saved and used to build other mashups (Req 3). The fact that mashups are designed for information integration (Req 1) and that they have already been used for analysis purposes in other domains [22], [23] motivated us to explore their applicability to check compliance rules. Throughout this paper we will describe how our approach addresses the 3 requirements as well as a methodology to use it. The framework has been validated with a detailed comparison to similar frameworks and by using it on real data.

The rest of the paper is structured as follows. Section 2 examines the state of the art on compliance checking, especially on compliance frameworks. Section 3 describes our mashup-based framework for forward compliance checking and exemplifies its use with a running example. Section 4 introduces a methodology for the use of the framework. Section 5 reports on our experience when applying it on real data. Finally, Section 6 presents the conclusions drawn from this work and outlines potential future work.

2 Related Work

A conceptual model of compliance checking similar to other conceptual models [1], [2] is depicted in Figure 1. A *Compliance Source* (e.g., a *Regulation* or an *Internal Policy*) defines a set of *Compliance Controls* that must be observed by the organisation. The compliance controls are not necessarily specific to a compliance source but may be shared by several of them. When the compliance sources are abstract, a consultant company is usually hired to define the compliance controls that apply to the organisation. Each control can be further decomposed into a set of *Rules* that specify how the controls can be materialised. This is usually dependent on the ISs used. For example, the rule *CR1* (cf. Section 1) would be defined differently if the checklist was created in a dedicated IS instead of being a document uploaded to a DMS. At design time, rules are assessed once for the whole process (e.g., one could check whether the organisation has a document specifying its security policies) but at run time, rules have to be assessed for each process instance (e.g., one could check that a given action has been recorded in an IS for each instance of the change request process). Moreover, rules have a *Process Context* in which they apply.

3. The complexity of a mashup lies within each component, and the greatest effort is put into how to integrate them.

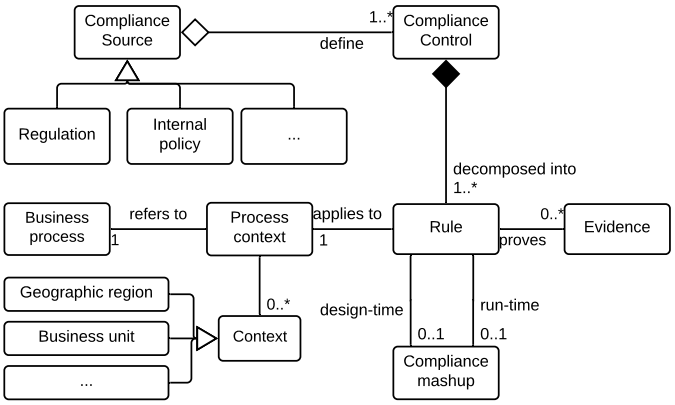

Fig. 1: Conceptual model of compliance checking

This includes the *Business Process* to which the rule refers and an optional *Context* within that process like a specific geographic region or a certain business unit of the company. Finally, there must be *Evidences*, like documents or log traces, that prove that the rules are being fulfilled (for *CR1*, the evidence would be the document uploaded to the DMS).

The main approaches addressing business process compliance checking up to 2010 were analysed according to several criteria in [24]. Later, Ly et al. derived 10 CMFs from a systematic literature review and several case studies in [5], which can be used as an evaluation framework of compliance checking approaches. The CMFs are divided into 3 groups: the *CMFs on modelling requirements* refer to the ability to deal with constraints that address aspects beyond control flow and they include *constraints referring to time* (CMF 1), *constraints referring to data* (CMF 2) and *constraints referring to resources* (CMF 3); the *CMFs on execution requirements* refer to requirements imposed by the execution domain in which compliance checking is performed and they include *supporting non-atomic activities* (CMF 4), *supporting activity life cycles* (CMF 5) and *supporting multiple instances constraints* (CMF 6); lastly, the *CMFs on user requirements* refer to aspects that improve the user experience and they include the *ability to reactively detect and manage violations* (CMF 7), the *ability to pro-actively detect and manage violations* (CMF 8), the *ability to explain the root cause of a violation* (CMF 9) and the *ability to quantify the degree of compliance* (CMF 10).

Ly et al. also classified compliance monitoring approaches into 5 categories [5]. One category includes *core compliance approaches*, which provide *specific techniques* for compliance checking with limited scopes. Most research efforts lie in this category. Regarding design-time compliance checking, many solutions separately model the business process and the rules, convert the 2 models into formal representations and apply model-checking algorithms to assess the degree of compliance [25], [26], [27]. These approaches especially focus on control-flow constraints and usually show the violations with counterexamples. Patterns are sometimes used either to simplify rule modelling or to facilitate compliance checking in scenarios that often recur [25], [27], [28]. Consequently, new pattern-based languages have been designed for the purpose of defining rules, such as PENELOPE [29] and BPMN-Q [27]. Other approaches annotate the process models using different formalisms for rule definition and analyse the annotated models for compliance checking [30], [31]. As far as run-time compliance checking is concerned, a detailed classification of approaches can be found in [5], including MobuconEC [32], which supports the monitoring of temporal and data constraints (among others) based on Event Calculus; MobuconLTL [33], which can monitor any rule that can be expressed in Linear Temporal Logic (LTL); and Seaflows [11], which uses Compliance Rule Graphs (CRGs) to monitor a stream of events encoded in a predefined format.

Another category of compliance checking approaches [5] includes *enabling technologies* like Complex Event Processing (CEP) or conformance checking. CEP can be very useful at run time in systems with a large amount of events and near real-time requirements but it cannot be used for design-time compliance checking. Moreover, having as input a stream of events prevents the execution of some checks, such as the verification of the content of a document. Concerning conformance checking, it differs from compliance checking especially in the fact that it involves checking the log against a complete process model instead of compliance rules.

The 3 categories that have received the least attention so far include, respectively: approaches that *collect compliance patterns* [34], [35], whose goal is to understand and categorise the different types of compliance rules that may exist, regardless of how to automate them; *approaches for specific domains* like healthcare [36]; and *frameworks* that provide general architectures or infrastructures for compliance checking [2], [16], [17], [18], [37]. The latter category is the most closely related to our work. We have found 5 frameworks that can be compared to ours. Next, we explain how the specifications of these frameworks support our 3 pursued requirements as well as the 10 CMFs.

Table 1 collects the results of the evaluation. In it, $+$ means that the feature is fully supported, $+/-$ means that it is partially supported, $-$ means that it is not supported, and $n.a.$ means that the criterion is not applicable (e.g., CMFs 4-8 are not applicable at design time). Note two considerations beforehand: (i) there are several reasons for a partial support of Req 2, as explicitly explained for each approach; and (ii) beyond being able to use exactly the same rule definition several times by copy-pasting it, Req 3 involves configurability (i.e., a definition can be adapted to be reused for different rules - e.g., by means of parameterisation).

The EU FP7 project COMPAS [37] worked on the development of an integrated solution for compliance checking [15], resulting in a framework for process monitoring based on CEP techniques. Three Domain Specific Languages (DSLs) were developed to define rules in 3 specific domains, and the processes were modelled with the View-based Modelling Framework (VbMF) [38]. The DSLs support control-flow, temporal and data-based rules, and their extensibility capabilities allow supporting the organisational perspective using the Role-Based Access Control (RBAC) model [39] (CMFs 1-3). The CEP engine is capable of aggregating and correlating low-level streams of events, so non-atomic activities can also be handled (CMF 4). We believe that dealing with activity lifecycles and multi-instance constraints is also possible thanks to CEP but we could not find evidence of such support in the COMPAS literature, so we consider

Approach	Req 1 integration	Req 2 variability	Req 3 reuse	CMF 1 time	CMF 2 data	CMF 3 resources	CMF 4 non-atomic	CMF 5 lifecycle	CMF 6 multi-inst.	CMF 7 reactive	CMF 8 pro-active	CMF 9 root cause	CMF 10 degree
COMPAS [15]	+/−	+/−	+/−	+	+	+	+	+/−	+/−	+	−	+	+
Awad et al. [16]	+/−	+/−	n.a.	+	+	+	+	+	+	+	−	−	−
eCRG [17]	+/−	+/−	+	+	+	+	+	+	+	+	+	+	+
OPAL [18]	−	−	+	+	−	−	n.a.	n.a.	n.a.	n.a.	n.a.	+	−
Schumm et al. [2]	−	+/−	+	+	+	+	n.a.	n.a.	n.a.	n.a.	n.a.	+	−
Our framework	+	+	+	+	+	+	+	+	+	+	−	+/−	+

TABLE 1: Requirements and CMFs supported by current compliance management frameworks

those functionalities partially covered (CMFs 5 and 6). CEP also enables continuous monitoring and early detection of compliance violations (CMF 7) but violation prediction is not a goal of the framework (CMF 8). Carefully designed Compliance Governance Dashboards (CGDs) [40] show many details of the compliance status (CMFs 9 and 10). As for our requirements, the framework checks compliance only at run time so Req 2 is partially supported. The DSLs are template-based, hence rule definition adaptation is enabled by customising the placeholders in the templates. However, each language is inherently linked to a domain, which hinders the actual reuse of the rule definitions outside a limited scope. Consequently, we consider this provides partial support for Req 3. The framework takes data from event logs and relies on the Esper engine for CEP but it does not communicate with other systems of the organisation. Req 1 is thus also partially supported.

Awad et al. introduced a framework for instant business process monitoring [16] that also makes use of CEP technologies taking into account the control flow, temporal constraints, data and resources (CMFs 1-3) and leveraging CEP advantages for considering non-atomic activities (CMF 4) divided into states (CMF 5) as well as multiple process instances executions (CMF 6). When a violation is detected, a recovery action is triggered in order to enforce compliance (CMF 7). Violation prediction and reasoning on detected violations are not addressed in the framework (CMFs 8-10). Req 2 is partially covered because design-time aspects are disregarded. The authors suggest using Semantics of Business Vocabulary and Business Rules (SBVR) for rule definition but the framework does not enforce a specific compliance rule language so it cannot be properly evaluated against Req 3. Finally, the framework integrates with the IT infrastructure that enacts the processes as well as with the event logs and the CEP engine but is not flexible enough to interact with other systems, partially supporting Req 1.

Knuplesch et al. developed a framework for visually monitoring business process compliance once the CMFs had been defined, targeting and addressing them all as described in [17]. The framework relies on the Extended Compliance Rule Graph (eCRG) language, an expressive graphical notation that enables the reuse of rule definitions (Req 3). Compliance checking is only done at run time (Req 2) with ad-hoc algorithms that assume that *all* the required input data are available in event logs. Therefore, the integration with further ISs of the organisation is not addressed (Req 1).

Unlike the previous approaches, the OPAL framework [18] targets design-time compliance checking. Business processes and rules are modelled with Business Process Execution Language (BPEL) and Business Property Specification Language (BPSL), respectively. These models are transformed to finite state machines and LTL, respectively, and a model-checking technique is used to find rule violations, which are shown to the user with counterexamples on the BPEL model (CMF 9). As BPSL allows the definition of Domain-Specific (DS) templates with configuration parameters, these can be reused for compliance rule definition (Req 3). However, the specification of constraints related to data and resources is part of the envisioned future work (CMFs 2 and 3). Because of this limitation and the restriction to design time, we consider Req 2 not to have been addressed. The integration with different process modelling tools and model-checking engines is possible but the framework does not access ISs of the organisation (Req 1).

Lastly, Schumm et al. developed a framework for modelling business processes that are likely to be compliant by design thanks to the reuse of *compliance fragments* (i.e., process fragments that are known to be compliant with specific rules) [2]. Process fragments modelled with Business Process Model and Notation (BPMN), BPEL or Unified Modeling Language (UML) are then formally represented in Reo [41]. LTL is preferred for rule definition, and control-flow, temporal, data and resource-aware constraints can be enforced (CMFs 1-3). However, Req 2 is only partially supported because run-time aspects are disregarded. A set of parameterised *compliance patterns* are specified for recurring rules, enabling rule definition reuse (Req 3). Model-checking tools are suggested for compliance checking so, like before, the counterexamples generated can help to understand the cause of the violations (CMF 9). Similarly to the OPAL framework, the integration with different process modelling tools and model-checking engines is possible but the framework does not access ISs of the company (Req 1).

The main difference between those approaches and our mashup-based framework is that we put specific emphasis on satisfying real needs that appear in industrial settings in which data are distributed among several heterogeneous systems (Req 1) and the large variety of rules urge putting in place configuration means that enable rule definition adaptation and reuse (Req 3). Furthermore, our framework supports design-time and run-time compliance checking and provides explicit mechanisms to add an open-ended set of checks (Req 2). Using appropriate formalisms, CMFs 1 to 6 could be supported thanks to the ability of the framework

to use different formalisms (cf. Section 3.3.1). CMFs 7, 9 and 10 could be supported based on an MCC system with our implementation of the BPCMS as described in Section 5.3. Violation prediction (CMF 8) cannot currently be supported.

3 Mashup-based Compliance Checking

We call *compliance mashups* the mashups used for the purpose of compliance checking. As depicted in Figure 1, they are used to provide a way to automate the checking of a compliance rule at design time and/or run time. Although several alternative compliance mashups may exist to check the same compliance rule, one of them has to be selected to implement the checking of the rule. If no mashup is selected, the rule will not be automatically checked.

3.1 Preliminaries: Fundamentals of Mashups

A mashup is a data-driven workflow (a.k.a. *data flow*) built with information from one or more data sources that might be transformed and propagated to produce a desired output in a reusable User Interface (UI) [19]. Mashups were developed to build new Web services or applications from existing data in an "easy" way, such that the end user did not need to have specific technical knowledge but only knowledge on the problem domain [20]. The original concept of *Web mashup* has evolved towards *data mashup* (for short *mashup*) and has already been used to address problems in a variety of domains, such as the analysis of molecular biology in bio-informatics [22] and the simplification of patient management in hospitals [23]. Intel Mash Maker, Yahoo! Pipes (now Pipes), IBM Mashup Center and Google Mashup Editor emerged as the first mashup builders to bring this technology to the end users. More recent and specialised mashup Application Programming Interfaces (APIs) can be found at www.programmableweb.com/category/mashups.

To show mashup appearance and use we have created the mashup in Figure 2. It returns the last 25 international pieces of news of 2 digital newspapers. Researchers could be interested in having a similar mashup in order to be kept up to date about their research interests (e.g., a mashup that automatically places on a map the venue cities where the next editions of their favourite conferences will take place). As illustrated in the figure, mashup editors allow the definition of the dataflow with 4 types of components. *Data sources* provide the information that shall be processed in the mashup and they range from data warehouses to Uniform Resource Locators (URLs) pointing at Really Simple Syndication (RSS) feeds or any kind of accessible data. The data sources in Figure 2 are the RSS feeds of the New York Times and The Australian newspapers. *Pipes* are the elements in charge of operating on data, so they all have inputs and outputs that represent the streams of data going in and out of the pipe, respectively. The input data they receive can come from another pipe component or from a data source, whereas the output data stream they produce can go to another pipe or constitute the output of the mashup to the *UI*. Finally, *data flows* represent the connections between pipes, data sources and the UI in a mashup.

Pipes can be General-Purpose (GP) components like those that handle collections of elements to sort or merge

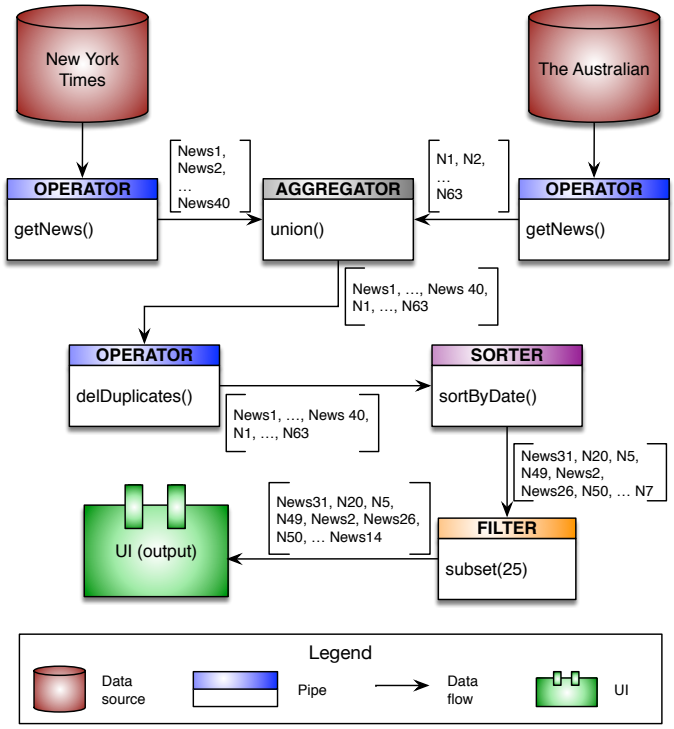

Fig. 2: Mashup collecting the last 25 worldwide pieces of news from 2 digital newspapers

them; and DS components that implement functions specific to the problem domain (e.g., handling geolocation data to enrich Google Maps with external information). Some frequently used pipes include operators, aggregators, sorters and filters. *Operators* extract, elaborate and transform the data, constituting a crucial part of the Extract-Transform-Load (ETL) process [42] from the input data source(s) to the output UI. They range from GP operators that implement well-known functions like *count*, *min* and *max*, to DS operators that, for instance, handle strings or extract and build geolocation information. In Figure 2, the operator *getNews()* extracts entries from an RSS feed, and *delDuplicates()* removes redundant entries from a set. *Aggregators* merge or group data according to some criteria. In Figure 2, *union()* merges entries from 2 sets into 1 set. *Sorters* return the input data in a specific order. In Figure 2, *sortByDate()* orders the entries in the input set by the value of the field *date*. *Filters* narrow down the flow of data, supporting the transformation of the information. In Figure 2, the pipe *subset(25)* retains only 25 items of the input set.

3.2 A Metamodel for Building Compliance Mashups

Each mashup framework, tool or specification proposes a slightly different mashup metamodel. Figure 3 depicts a metamodel that includes all the elements that are necessary for building compliance mashups. A mashup is composed of *Nodes* (which can be *Data Sources* and *Pipes*) and *Data Flows*, with the same meaning as defined in Section 3.1. The pipes and data sources used are referred to by their *name* and are chosen from a catalogue of *Pipe Components* and *Data Source Components*, respectively. The characteristics of the input and output data streams of the components

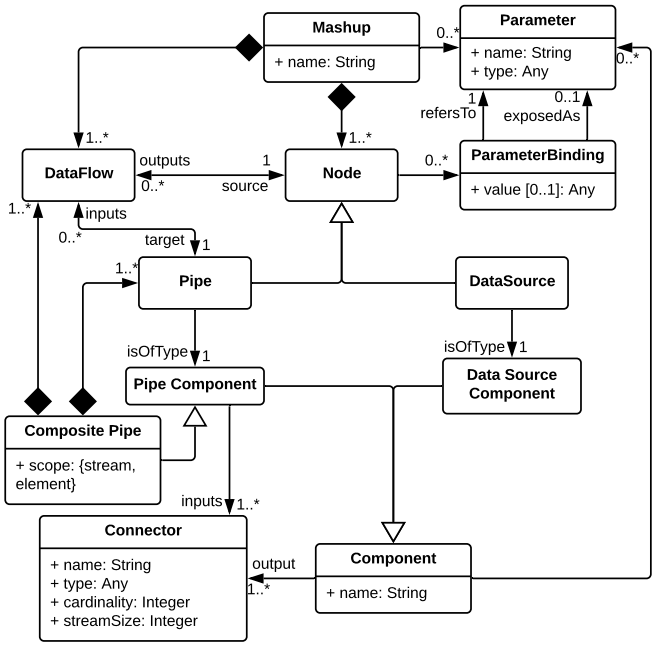

Fig. 3: Metamodel for building compliance mashups

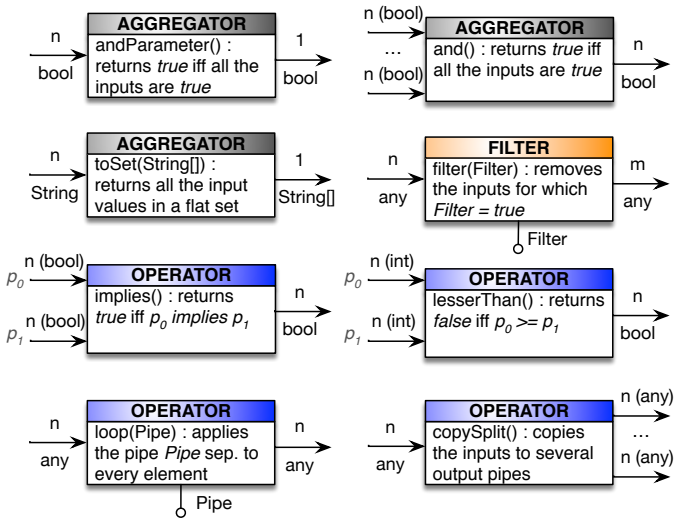

Fig. 4: Example GP pipe components

are defined by means of *Connectors*. Each connector has a *name*, a *type* that specifies the type of the data carried in the data stream, a *cardinality* that indicates the number of streams that go through the connector, and a *stream size* that indicates the expected number of elements in each of the data streams. The stream size of most connectors is n, meaning that they can carry data streams of any length. However, some connectors may have a stream size of 1 (e.g., a connector to specify the output of a component that counts the number of elements received in its input stream). Connectors are also used to specify the characteristics of the output streams of data source components.

Figure 4 shows several pipe components that can be used in a compliance mashup. Each pipe is represented

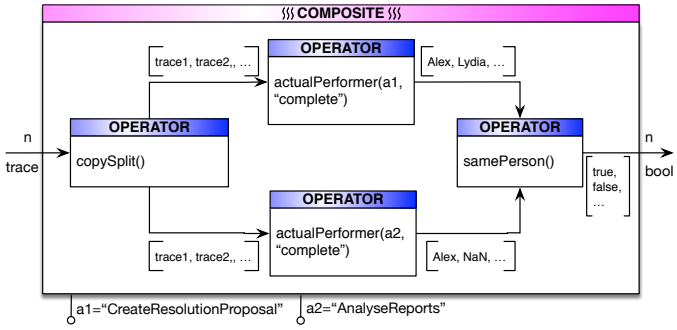

Fig. 5: Composite pipe to check binding of duties at run time

with a box that includes the pipe type, its name and a small description of its functionality. The connectors that specify their inputs and outputs are the arrows going in and out of the box, respectively, and their stream size and type are specified above or under the arrow. Tags (p_i) are used in pipes in which the input order matters (e.g., *lesserThan()*).

Pipes and data sources may have *Parameters* to customise their behaviour (e.g., one can have a generic filter pipe whose filtering condition is specified by means of a parameter). In Figure 4 the parameters are specified as lollipops at the bottom of the box. Mashups must include a *Parameter Binding* for the parameterised components. These parameter bindings either provide specific values for pipes and data sources or expose those parameters as parameters of the mashup itself. This allows the external *configuration* of the mashup and hence, enables the reuse of mashups in different scenarios. Concerning composite pipes, they are a special type of pipe component aimed at creating new higher-level pipes based on the composition of lower-level pipe components. They are useful because of two reasons: (1) they enable abstraction, making complex mashups easier to understand; and (2) they facilitate the reuse of a complex structure of pipes in different mashups. Like mashups, a composite pipe groups several pipes connected with data flows and includes parameter bindings for the parameterised pipes it contains. However, unlike mashups, composite pipes cannot contain any data source. Instead, their inputs and outputs are determined by the pipes contained in them that do not have any input or output data flow, respectively. Finally, the *scope* of a composite pipe can be of type *stream* or *element*. In composite pipes whose scope is of type *stream*, the input is processed as a whole and their behaviour is the same as if their components were used in the mashup directly. In case of a scope of type *element*, the composition is applied to each element in the input stream separately (one by one). Graphically, the latter composite pipes have the component name surrounded by a three-waves symbol. To showcase the difference let us suppose that we have a composite pipe that contains just one pipe component that counts the number of elements received in the input data stream. If the scope of the composite pipe is of type *stream* and its input stream contains 5 elements, then its output stream contains just one element: the number 5. Instead, if the scope is of type *element*, its output stream contains 5 elements, each of them being the number 1.

An example of a composite pipe with a scope of type

element is depicted in Figure 5. It receives *n* executions of a business process (traces) as input and 2 specific activities that are needed to perform the checks as parameters. The *copySplit()* operator copies the stream into 2 output flows, each of which serves as input of an *actualPerformer(act,state)* operator. This operator returns the resource allocated to activity *act* when the activity reached the state *state* for every trace[4]. The *samePerson()* operator checks whether the items in the 2 input streams are the same on a 1:1 basis, returning a stream of boolean values that is the output of the composite pipe. The true values indicate a binding of duties between the 2 given activities of the business process, meaning that they were allocated to the same resource [43].

In short, a metamodel suitable for building compliance mashups must include (i) mechanisms to enable the reuse of (parts of) mashups (i.e., composite pipes and parameters); and (ii) the ability to define the scope of a composite pipe to cover *streams* (useful at design time to provide a global evaluation of the rule) and *elements* (useful at run time to provide an evaluation of the rule for each process instance).

3.3 Compliance Mashups in Practice

To illustrate the use of compliance mashups in practice, we use a real process frequently utilised in the Andalusian Public Administration (APA), which serves to more than 8 million end users. The process represents the procedure to create and process a resource resolution proposal for hiring people. Figure 6 shows its BPMN model. Once a resource resolution proposal is created at the Management Department (MD), it is concurrently evaluated by the Consultative Board (CB) and the Legal Department (LD). When both evaluations are finished, the MD analyses the generated reports and decides whether an external resolution is required. In case it is, a request is sent to an external committee, which must create and send a new resolution. Otherwise, the resolution proposal is reviewed and changes are applied to the initial document. In any case, the final resolution is signed and archived, and the result is appropriately notified.

Figure 7 depicts the excerpt of the organisational model related to the process, which comprises 3 organisational units and 8 positions within them occupied by 11 people[5]. The positions are hierarchically structured. Figure 6 also illustrates the position-based resource assignments of the process activities. The assignment of several positions to an activity means that the resources occupying any of them can execute it (e.g., activity *Review resolution proposal* can be done by a technician of any of the 3 units). More complex assignments have been omitted in the figure for the sake of readability. Specifically, there is a binding of duties between *Create resolution proposal* and *Analyse reports* indicating that these activities must be executed by the same resource. Several approaches could be used to define such constraints, either textually (e.g., [43]) or graphically (e.g., [44]).

As part of the compliance controls that affect this process, the Business Manager of the MD (the process owner) needs to know whether the following rule is met: "If the resolution proposal has been created and the respective reports from the CB and the LD are available, then such reports must be eventually analysed by the same person who created the resolution proposal" (*CR2*). This rule involves checks related to several process perspectives, specifically: (i) that specific data objects (*Report CB* and *Report LD*) are created (informational perspective); (ii) that one activity (*Analyse reports*) is executed when certain conditions are met (in this case eventually after the execution of *Create resolution proposal* and the activities writing the two data objects) – behavioural perspective; and (iii) that the binding of duties between two activities (*Create resolution proposal* and *Analyse reports*) is fulfilled (organisational perspective).

Next, we detail the considerations for the design of the mashup components and the mashups themselves to check *CR2* at design time (Figure 8) and at run time (Figure 9).

3.3.1 Designing the Mashup Components

In a compliance mashup, the data sources access the repositories and ISs where the organisation stores data that is relevant to the evidences. Examples of data sources that may exist are the repositories where business process, organisation and data models are stored; the DMS of the company necessary to check which documents have been created; or the Enterprise Resource Planning (ERP) system that provides access to financial information[6]. In our example, we are checking the compliance of a resource-aware process model (i.e., a model enriched with resource assignments) at design time, and a set of running process instances at run time. Therefore, the data sources are a repository of business process models and a log of running process instances.

Regarding the pipe components, GP and DS pipes might be necessary to manipulate the data coming from the data sources. Figure 4 depicts representative examples of GP pipes. More GP pipes can be found in mashup development frameworks, tools and specifications [19], [20], [21]. The set of available DS pipes depends on the kinds of checks to be performed in a specific domain. Some might be specific to one organisation or even to a process and cannot be reused (e.g., a pipe that parses an ad-hoc document to retrieve the subprojects contained in one particular project). However, many DS pipes can be reused in different processes and/or organisations. For instance, one may have DS pipes that implement operations on BPMN models, such as obtaining all the activities defined in a process or all the resources assigned to an activity; or operations to deal with event logs, such as checking for precedence between 2 activities.

The DS pipe components can be implemented in an ad-hoc manner or encapsulate existing solutions (cf. Section 2). If an existing technique is going to be encapsulated, the granularity of each pipe must be decided. One could have a coarse-grained pipe that encapsulates the whole technique (e.g., a MobuconLTL [33] component that uses an LTL specification as a parameter to check the LTL formula against the process instances in an event log); or several fine-grained pipes that implement specific steps (e.g., use MobuconLTL to implement several fine-grained pipes that carry out different checks that usually appear while monitoring the compliance of event logs, such as the occurrence

4. NaN is used for any undefined value.

5. Their names have been changed for privacy reasons.

6. Note that the data sources do not need to provide access to the whole ERP system but they can be as specific as necessary.

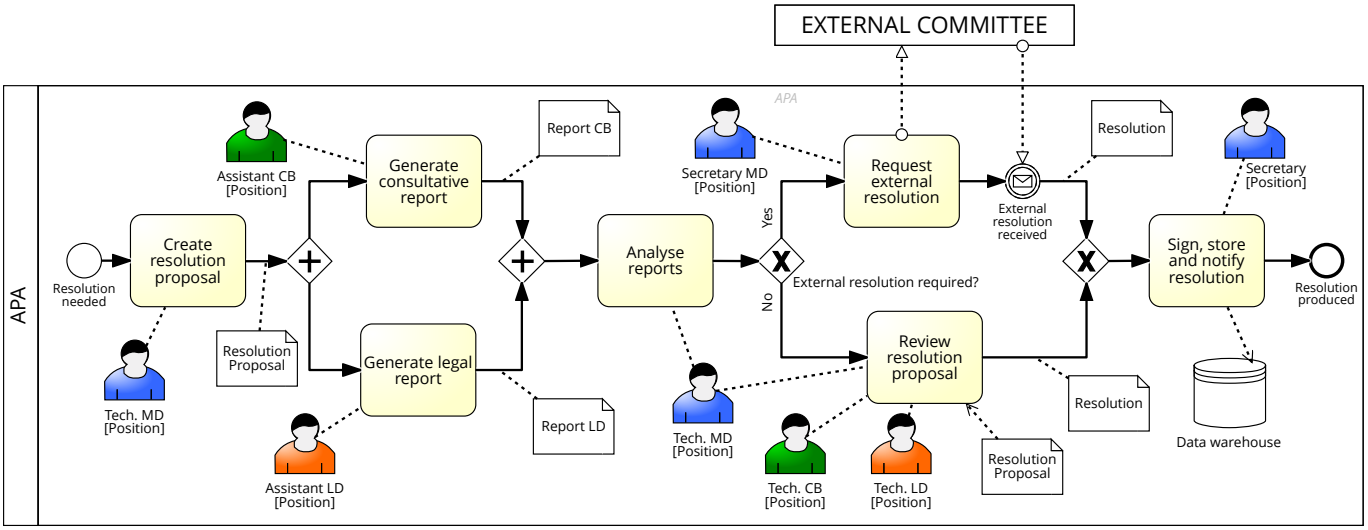

Fig. 6: Simplified version of the process to generate a resource resolution proposal

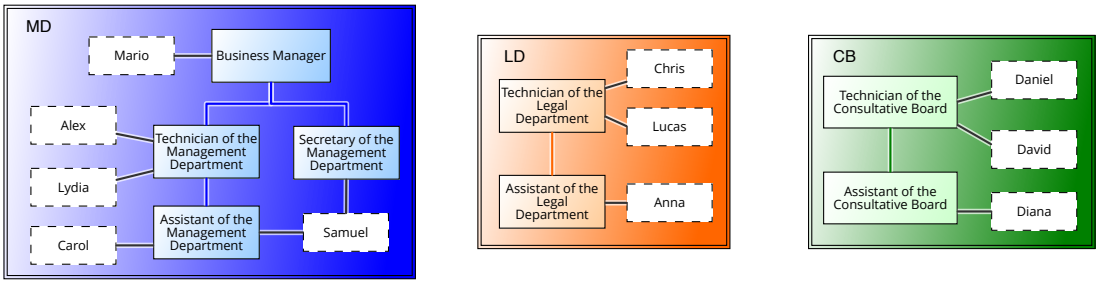

Fig. 7: Excerpt of the organisational model of the APA

and co-occurrence of activities, or the ordering of execution between activities; in this case, the parameters of the pipe would be the activities instead of a whole LTL specification).

The choice between using coarse-grained or fine-grained pipes depends on several factors. Coarse-grained pipes have the advantage of being able to exploit the logic languages in which they are usually built to optimise or reason about the checks in a manner that cannot be done if the logic of the compliance check is in the mashup. Instead, fine-grained pipes have the advantage of making the logic of the compliance check more explicit in the mashup and allowing combining different techniques to implement the compliance checks. For instance, data-based checks could be implemented using [32], whereas temporal checks could be implemented using [11]. An example of this can be found in Figure 8, which combines one technique for checking resource assignments (*bodDT(om,a1,a2)*) with another for checking control-flow constraints (*alwaysFollows(bp,a2)*). Moreover, fine-grained pipes are also useful if the same type of check is used in many mashups since it avoids that the same customisation be spread through several mashups.

Finally, besides combining different techniques, DS pipes can also be used to seamlessly integrate data from different ISs. For instance, if some data are not stored in the event log but in a document saved in a DMS, one could implement a pipe (or a composite pipe) that extracts the name of the document from the log, retrieves it from the DMS and checks its content. Figure 9 shows an example of this with components *getUrl(D[1])*, *getDocument()*, and *lastModified()*.

3.3.2 Composing the Compliance Mashup

Using 1 data source, 3 GP pipes and 4 DS pipes, and receiving the ID of the resource-aware business process model to be checked (*bp*), the organisational model associated with the process (*om*), two activities (*a1* and *a2*) and a set of data objects (*D*) as input parameters, the check of *CR2* at design time can be done in the following steps (cf. Figure 8):

1) The model of the business process *bp* is extracted from the repository of business process models and copied into several streams to be processed by different pipes.
2) The activities of the process that write the input data objects are identified in the model since, according to *CR2*, both the activity *a1* and these activities must be finished in order to check the rest of constraints. The second parameter of the *writes(D,any)* pipe used for this purpose indicates whether the activities that write *any* or *all* the data objects *D* are selected.
3) A set is then created with the activities previously identified plus *a1*.
4) The next step is to check that the input activity *a2* always follows the activities in that set as defined in the model *bp*. The result of that check is a boolean value.
5) The last constraint to be satisfied is the binding of duties between *a1* and *a2* also as specified in the process model *bp* against the organisational model *om*.

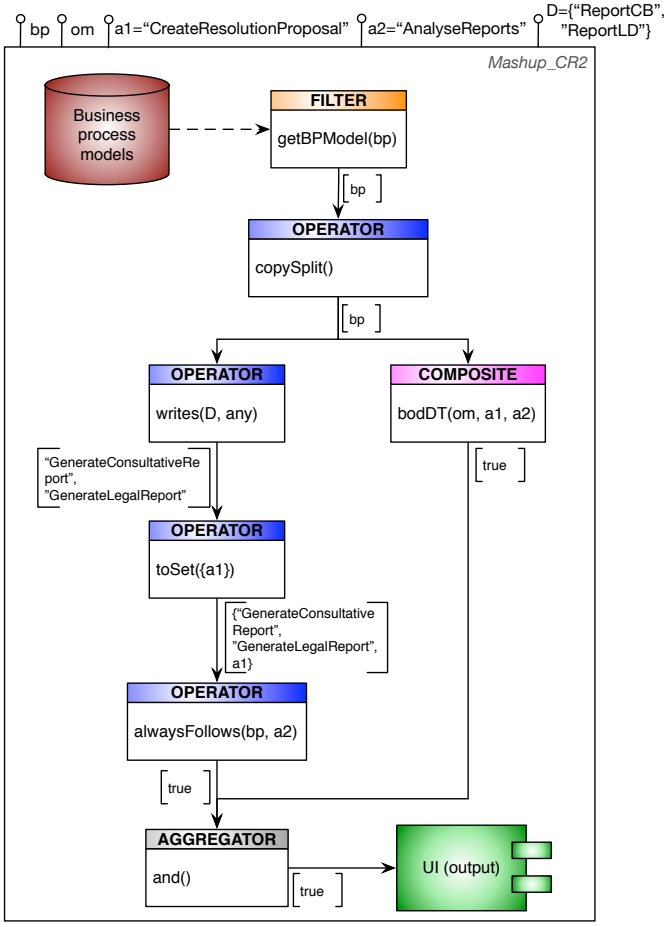

Fig. 8: Compliance mashup to check rule CR2 at design time

Case	Activity	Time	Resource	Type
1	Create resolution...	2019-05-21T08:40...	Alex	complete
1	Create legal rep...	2019-05-22T14:03...	Anna	complete
1	Create consulta...	2019-05-23T11:20...	Diana	complete
1	Analyse reports	2019-05-25T14:51...	Alex	complete
2	Create resolution...	2019-06-11T09:15...	Lydia	complete
2	Create consulta...	2019-06-27T15:41...	Diana	complete
...	...	...	...	...

TABLE 2: Event log of running process instances

6) Finally, the two boolean values obtained from (4) and (5) are aggregated with an *and()* pipe. The output indicates whether the setting is prepared to fulfill *CR2*.

As depicted in Figure 8, in this concrete case the process is compliant with *CR2* at design time. If we wanted to give a slightly different meaning to *CR2*, we could reconfigure the properties of the respective components in Figure 8 and reconnect them, or even insert new components to deal with the new interpretation (e.g., if the activity *Analyse reports* should *immediately* follow the activity *Create resolution proposal*).

We assume that for every process there is a data source that provides an event log with ongoing instances (cases), where every event contains information about an activity, its state (where at least the transition type *complete* is stored), the temporal data specifying when it was stored, and an

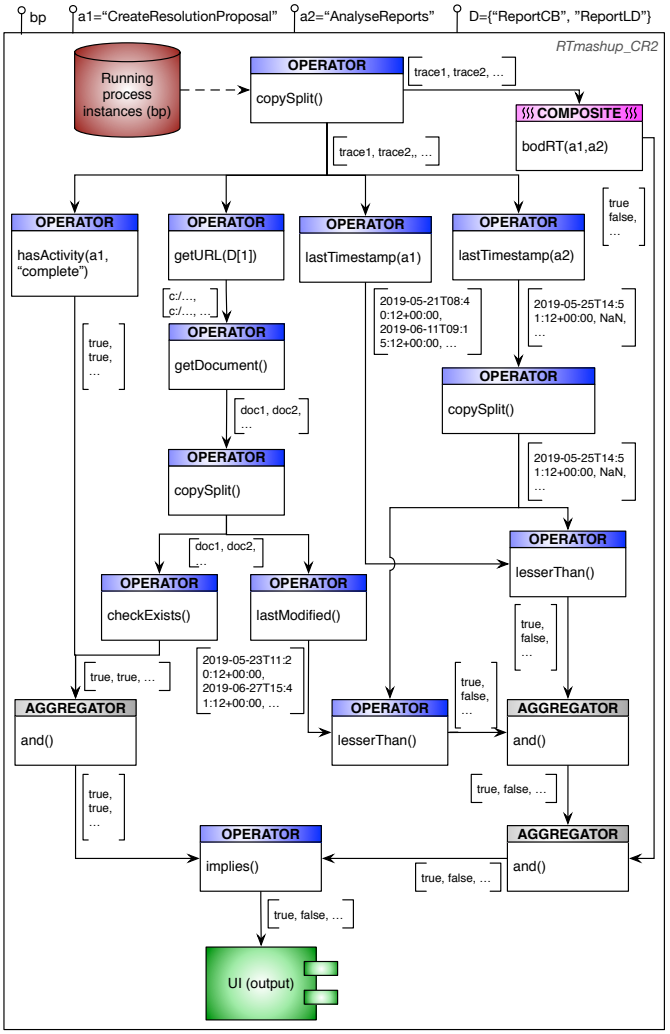

Fig. 9: Compliance mashup to check rule CR2 at run time

associated resource. This is common information stored in real-life event logs [45]. An excerpt of an event log for the example scenario is shown in Table 2. Then, the check of *CR2* at run time can be done in the next steps (cf. Figure 9):

1) The traces of the running instances of the process *bp* are extracted from the respective event log and copied into several streams to be processed by different pipes.
2) The first check that has to be done (in every trace) is that activity *a1* has been completed.
3) Moreover, the 2 input data objects *D* must have been created. That implies obtaining the location of such files and retrieving them from the respective repositories. Note that for the sake of readability, only the operations corresponding to 1 of the data objects are illustrated. The respective components should be duplicated.
4) Several checks will be performed upon those documents, so the files are first copied into several streams to be processed separately.
5) To check the first part of rule *CR2*, activity *a1* must have been finished and the documents must exist in the repositories. An *and()* pipe is in charge of that.
6) The second part of the rule states that activity *a2* must be executed after *a1* is completed and the doc-

uments are written. *lesserThan()* pipe components check whether an instance of activity *a2* has been completed after the last modification of the data objects *D* according to their timestamps. Similarly, another *lesserThan()* pipe checks whether the last instance of *a1* was completed before the last instance of *a2*. These checks are aggregated with an *and()* pipe to ensure the control-flow constraint.

7) The result of (6) is then aggregated with the result of the binding of duties check. The latter is conducted by a composite pipe (*bodRT(a1,a2)*) whose implementation is shown in Figure 5.
8) Lastly, the *implies()* pipe checks whether there is an implication relation between the result of (5) and the result of (7). The outcome of this check determines whether the rule is fulfilled or not.

As depicted in Figure 9, the first running instance of the process is currently compliant with *CR2* but the second instance currently violates the rule because there is no evidence about the subsequent completion of activity *Analyse report*. Note that a firm statement about the fulfilment or violation of a rule can only be made once the execution of the process instance is over. The intermediate compliance results could change depending on the occurring events.

4 A Methodology to Implement Mashup-based Compliance Checking

Several roles and documents are involved in the process of creating a mashup-based compliance checking (MCC) system. Figure 10 provides an overview of how such a process can be carried out. We assume that a previous process has identified which are the regulations that apply to the organisation as well as the internal policies that must be implemented according to good practices or corporative business goals. Therefore, the starting point is the set of regulations and internal policies that must be accounted for by the MCC system. Next, we detail each of the steps.

Create compliance controls. The first step is to create the controls to be satisfied. The amount of effort required by this task highly depends on two factors. On the one hand, the level of abstraction can vary significantly between regulations. For instance, the Payment Card Industry Data Security Standard (PCI DSS), which is an information security standard for organisations that handle credit card data, is very specific and already provides a catalogue of controls that must be supported; whereas other regulations like Sarbanes-Oxley Act (SOX) are more generic and a greater effort is required to identify the set of controls. On the other hand, there is usually some overlapping in regulations. This means that additional effort should be put into trying to create controls that are valid for several regulations at the same time because keeping the number of controls as low as possible will make it easier to deal with the complexity of the system. There are some methodologies that might be useful in this step (e.g., [46]). The outcome is the catalogue of controls that should be satisfied by the organisation.

Identify compliance rules. The second step is to identify how these controls can be materialised within the organisational context. As explained in Section 2, this requires a deeper understanding of the processes and how the ISs are organised and hence, a cooperative effort between compliance experts, process owners and system architects. *CR1* (cf. Section 1) is an example of rule for *CC1*. The outcome of this step is a set of rules that detail how to check each of the compliance controls at design time and run time.

These first two steps are necessary to build an MCC system but they are not yet related to mashups in any way. In fact, these steps (possibly with minor variations) are common to all compliance checking system approaches that have been discussed in the literature [1], [2]. It is after the compliance rules have been identified that the mashup creation process starts. This is specified in the subprocess depicted in Figure 10 and comprises the following steps.

Identify mashups and their DS components. This step takes as input the controls with rules and obtains both the definition of the mashups that are necessary for specifying the compliance rules and the DS pipes that must be implemented. Regarding the former, in Section 3 we showed how mashups can be parameterised to be reused by several rules. Thus, in this step the goal is to group all the rules that have some similarities. For instance, *CR1* should be grouped with the rules that involve periodically checking that a given document has been uploaded to the DMS. Concerning the DS pipes, their identification requires a compliance mashup expert supported by a system architect that knows the details of the ISs of the organisation. For insights on how to build high-quality mashups we refer to [47].

Implement DS components. This task involves developing the DS components of the mashups from the list of DS pipes previously obtained. It should ideally be done by developers that are knowledgeable both on the compliance mashup framework and on the technologies with which the DS pipe components must interact (e.g., the DMS).

Develop mashups. The next step is to build the mashups as detailed in Section 3.3. While building the mashups it is recommended to group small parts in composite pipes because the abstraction they provide makes complex mashups easier to understand, and they increase the reuse possibilities. This should be done by a compliance mashup expert.

Assign mashups to rules. The last step involves assigning one mashup to each compliance rule. Concrete values are given to the parameters of the mashup. This step can be done by a compliance mashup expert.

Note that it is usually hard to identify all the mashups and DS components that shall be necessary in a first attempt. Several iterations of the mashup creation process may be necessary to build mashups for all the compliance rules.

5 Experience Report

Our approach was used to build a BPCMS [8] based on an MCC system for the IT department of a multinational company from the energy supply domain as an R&D project.

5.1 Context

The goal of the BPCMS was to address compliance checking at design time and at run time for the IT projects in which the company was involved. The company had

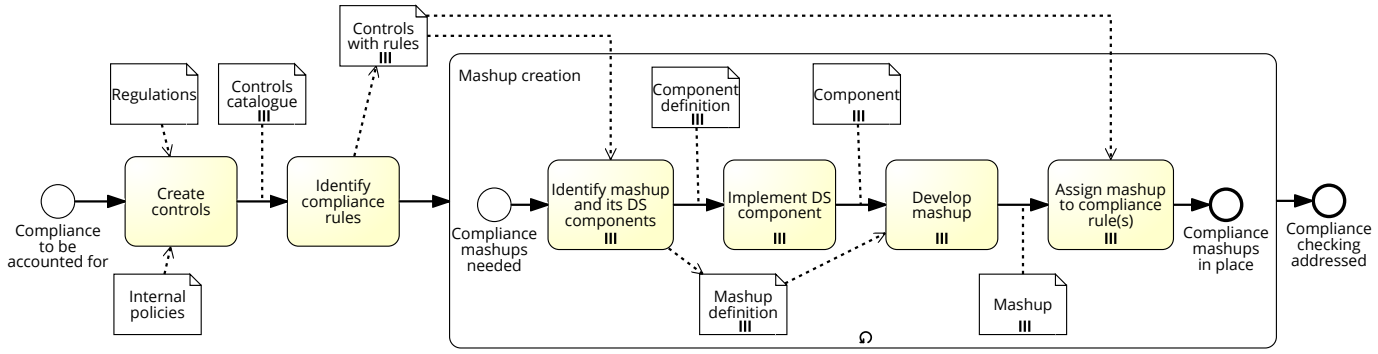

Fig. 10: Creation of an MCC system

recently finished a process modelling and analysis effort for that department, resulting in the definition of more than 20 processes that specified how the projects should be carried out. They had also defined 122 compliance controls that were obtained by a consultancy firm after analysing the implication of 3 well-known regulations (SOX, SCIIF and L262) plus additional controls that responded to internal business policies. An example of control is "For new projects, the project owner must verify the correct execution of the Test Plan". At the moment of starting the project, these compliance controls were not being evaluated yet. Instead, a previous set of compliance rules was checked by the compliance department. The assessment of the rules depended on the application that contained the information required as evidence. For some applications a rule was evaluated automatically but for others there was no automation and it had to be checked manually. Besides, since the way rules were implemented varied from rule to rule, it took time and resources to consolidate them in a unified view.

Thus, the goal of the project was threefold: (i) to implement the new set of controls defined by the consultancy firm; (ii) to automate many of the compliance rules that were checked by hand at that moment; and (iii) to automatically consolidate all compliance rules that were checked in different tools into one unique tool with a common dashboard and automatic alerts sent to the appropriate person when a rule was not fulfilled. They were interested in doing so in a way that new controls could be easily implemented or removed since the set of compliance controls was expected to change due to the need to support new regulations because of recent acquisitions in different countries.

5.2 Application of the Approach

To apply our approach we first developed an MCC engine in .NET, which included 4 GP pipe components, a mechanism to model mashups based on the software modelling tool Enterprise Architect, and a mechanism to specify controls and link them with mashups, which was based on an Excel spreadsheet. The MCC engine was completely domain-independent and could be reused in any project.

After developing the MCC engine we followed the methodology described in Section 4 to apply our framework in the context described above. Since the controls catalogue was already defined, we skipped the first step of the methodology and started with the identification of compliance rules. This task was performed by 1 researcher and 2 members of the compliance department of the company in three 2-hour meetings with some preparatory sessions of about 8 hours that were held along 2 weeks. The result was a description of the rules that had to be checked. Each control had 2 rules associated to it (1 for design-time compliance checking and 1 for run-time compliance checking) except for 3 cases in which 2 or 3 rules were defined for each of them. The reason why different rules were defined for design time and run time is that they had no explicit traceability between the process models and the ISs that supported them. Hence, it was not possible to automatically infer how to implement a run-time rule from a design-time rule because it required implicit knowledge on the IS architecture.

The next step was the identification of the mashups and the DS components of the mashups that were necessary to check the rules. This was carried out by 2 researchers and members of the teams that supported the ISs used in the BPCMS. We first categorised the rules obtained in the previous step. This resulted in 11 different categories. The distribution of rules amongst these categories was not homogeneous. Some categories had up to 38 rules while others had just 1. The categorisation criteria we used included the IS involved (e.g., DMS, model repository, or helpdesk system), the type of check (e.g., the existence of a document, an element in a model, or specific data in an IS) and the conditions under which the check applied (e.g., after the execution of an activity, or when a project is critical). Once the compliance rule categories were identified, a mashup and the necessary DS components were devised for each of them. They included both data sources and pipes. Concerning the former, only 1 data source was defined for each IS that was used in the mashups except for 1 case in which 2 data sources were defined because the system handled very different data types. As for the pipes, we chose to use fine-grained pipes because the same pipe component was going to be used in many mashups and we wanted to avoid the same customisation to be spread over several different mashups. The next steps involved the implementation of the data sources and DS pipes as well as the mashups. This step was performed by 2 researchers and 2 software developers, who implemented the components that involved interacting with external ISs. Finally, the mashups were assigned to controls using a spreadsheet that linked each control to 1 or more mashups and provided their parameters.

All these steps, from the identification of the mashups to their implementation and assignment to controls was an iterative process because some mashups required unforeseen components that had to be defined or because some components were generalised so that they could be used in other mashups. After all the iterations a total of 11 mashups based on 5 data sources and 18 DS pipes (9 filters, 4 to perform API calls to the IS, 3 to extract information from complex data structures, 1 URL builder, and 1 to query inside documents) were defined. The definition and implementation of mashups and DS components that did not involve the interaction with external ISs took around 60 hours, whereas the implementation of the components that connected the MCC system with external ISs was done in 30 hours by a team that was already familiar with those ISs.

Figure 11 depicts the system we obtained. 4 different ISs provided data to the mashups for compliance evaluation, namely: process models from an enterprise modelling tool for design-time rules, project and process documents from a DMS, data about projects status from a project management tool, and data about incidents and claims from a helpdesk system. The information obtained from the compliance evaluation was exploited in 2 different ways. On the one hand, compliance violations were notified by email to the person responsible for the project or process that caused the violation. On the other hand, a dashboard was developed to present the compliance evaluation results. The dashboard was structured in 3 levels of detail: *executive level*, which provided an overview of the state of compliance grouped by normatives and organisational structure; *management level*, which provided more details about the state of the compliance in the different functions of the organisation; and *operational level*, which provided full details about the state of each of the controls involved. Based on the information provided by the MCC engine, the dashboard and the notification system was implemented in 45 hours.

Due to data privacy reasons, we cannot share the full working system, but the source code of the MCC engine is available at https://github.com/isa-group/mcc-engine.

5.3 Reflections and Lessons Learned

The application of our conceptual framework in practice helped us to validate its feasibility and usefulness as well as to refine the methodology. It showed that it was possible to model and check real design-time and run-time compliance controls that involved 4 ISs already deployed in the company. Furthermore, we could access the possibility of supporting CMFs 7, 9 and 10 (cf. Table 1). Since rule violations can be detected during process monitoring and reported to the person(s) responsible for the rules as well as shown in a compliance dashboard, subsequent recovery actions could be taken (CMF 7). Moreover, thresholds can be defined in order to classify the degree of compliance both at process-instance and at process level (CMF 10). The reason of a violation cannot be specifically reported but the mashup component(s) that failed can be identified and hence, we can figure out where the problem is (CMF 9).

We also learned some lessons concerning several steps of the methodology and the approach, as described next:

1) The identification of compliance rules is a key activity in the methodology and may take more time than initially expected. When the project started, it seemed that having the controls catalogue was enough to start implementing the MCC engine. However, it turned out not to be true. There is usually a gap between a description of a control and the rules in which it can be decomposed. Bridging this gap is not straightforward and often requires knowledge on both the control and the way information is handled by the ISs.
2) The parameterisation of mashups happened to be very useful as many rules were based on the same mashup and only changes in the parameters were necessary. As a matter of fact, it was necessary to define just 11 different mashups to implement the compliance rules for all controls. However, it was necessary to make an effort in order to make these mashups as reusable as possible, which sometimes required a couple of iterations.
3) The integration of new ISs with the MCC engine was easier than expected. The reason is that data sources and DS pipes provide useful abstractions that, on the one hand, help the developer to focus just on the interaction with the IS without considering the specific compliance controls and, on the other hand, provide a homogeneous way to interact with external ISs.
4) Some compliance rules may require human intelligence to ensure that, e.g., the content of a document adheres to certain guidelines that could not be automatically checked. In our project, these rules were loosened and the mashup checked that a specific section appeared in the document regardless of its content. An extension could consist of introducing a new pipe that handled the compliance checking interaction with a human user (e.g., through emails or a dedicated UI). For each document that had to be checked the component could return one of these three states: valid, invalid, and pending. The last state would allow the system to provide data without having to wait for the user validation.
5) The specification of the controls catalogue, the compliance rules and the parameterisation of the mashups should be done together and in a way that is accessible for end users, not only for system developers. This allows end users to change them more easily. In the project we used an Excel spreadsheet for that task because it was consistent with the way in which the controls catalogue was handled before. This had some advantages, such as the fact that it was familiar to the users. However, it was also prone to errors because the system could not check the correct parameterisation of the mashups until they were deployed.
6) Debugging the system, especially the mashups, was not an easy task due to their componentisation. It would be convenient that the MCC engine provided some sandbox to help to debug mashups. Nonetheless, the componentisation in data sources and pipes eases the creation of unit tests of the different components.

6 Conclusions and Future Work

The development of a BPCMS that had to be used in a real scenario involving distributed and heterogeneous ISs and a catalogue of 122 diverse compliance controls led us to focus on needs that were not fully covered by existing compliance

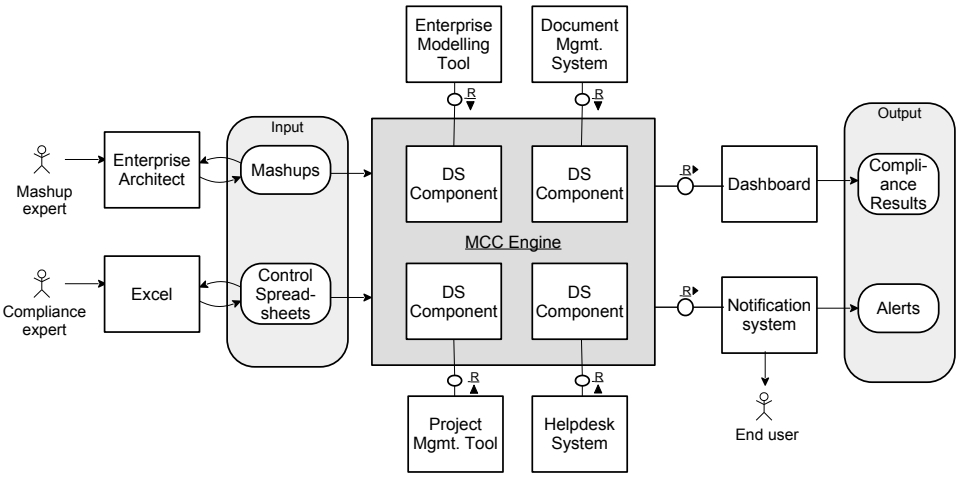

Fig. 11: Architecture of the BPCMS developed

proposals. We chose to develop a mashup-based approach encouraged by the advantages found in this technology from their use in Web applications [20].

From the work performed we conclude that compliance mashups provide a framework that may help to simplify an important problem in business process compliance management: the specification of compliance rules in a way that enables their automated checking. Its power is propelled by the fact that already existing approaches can be utilised for specific functionalities provided that they are adapted to be used together. In that regard, the framework does not aim to overlap with other compliance checking approaches but to complement them by adding an integration mechanism. In addition, it moves the state of the art on business process compliance closer to complex real-life settings. New components could be added to the framework based on new system requirements (e.g., to cover new sets of rules). Similarly, note that although the focus of this paper is on forward compliance checking, the idea could be applied to extend the conceptual framework to address backward compliance checking, too.

Our framework is best suited for organisations with distributed and heterogeneous ISs and/or that manage a significant amount and variability of compliance rules, which can benefit from the open-ended set of types of reusable rules. For organisations with homogeneous process-aware ISs and few compliance rules, it is probably easier to use a solution that focuses on those specific data sources and types of rules, like those covered in [5]. Likewise, if processes are executed by hand or process models are not available, the automation capabilities of the framework are rather limited. Finally, for other types of compliance like IT security compliance it could be useful as long as its compliance rules can be expressed as a data flow in a mashup.

As a next step, compliance prediction features could be incorporated and the application of compliance mashups in different domains could be explored to check on generalisability and scalability. A similar mashup-based approach could also be used for other types of analyses in the BPM field, leveraging their reusability and integration properties.

ACKNOWLEDGEMENTS

This work was partially funded by the Austrian Science Fund (V 569-N31); projects US-1264651 and P18-FR-2895, and the MCI/AEI/FEDER, UE (RTI2018-101204-B-C21, RTI2018-101204-B-C22 and RTI2018-100763-J-100).

REFERENCES

[1] M. Papazoglou, "Making Business Processes Compliant to Standards & Regulations," 10 2011, pp. 3 – 13.
[2] D. Schumm, O. Türetken, N. Kokash, A. Elgammal, F. Leymann, and W. van den Heuvel, "Business Process Compliance through Reusable Units of Compliant Processes," in *ICWE Workshops - Revised Selected Papers*, vol. 6385, 2010, pp. 325–337.
[3] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management - Second Edition*. Springer, 2018.
[4] M. E. Kharbili, A. K. A. de Medeiros, S. Stein, and W. M. P. van der Aalst, "Business process compliance checking: Current state and future challenges," in *Modellierung betrieblicher Informationssysteme (MobIS) Conference*, 2008, pp. 107–113.
[5] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. van der Aalst, "Compliance Monitoring in Business Processes," *Inf. Syst.*, vol. 54, no. C, pp. 209–234, 2015.
[6] S. C. Tosatto, G. Governatori, and P. Kelsen, "Business Process Regulatory Compliance is Hard," *IEEE Trans. on Services Computing*, vol. 8, no. 6, pp. 958–970, 2015.
[7] S. Sadiq and G. Governatori, "Managing Regulatory Compliance in Business Processes," in *Handbook on BPM 2: Strategic Alignment, Governance, People and Culture*, J. vom Brocke and M. Rosemann, Eds. Springer, 2015, pp. 265–288.
[8] C. Cabanillas, M. Resinas, and A. Ruiz-Cortés, "Exploring Features of a Full-Coverage Integrated Solution for Business Process Compliance," in *CAiSE Workshops*, vol. 83, 2011, pp. 218–227.
[9] A. Gericke, H.-G. Fill, D. Karagiannis, and R. Winter, "Situational method engineering for governance, risk and compliance information systems," in *Int. Conf. on Design Science Research in Information Systems and Technology (DESRIST)*. ACM Press, 2009, p. 1.
[10] C. Giblin, S. Muller, and B. Pfitzmann, "From Regulatory Policies to Event Monitoring Rules: Towards Model-Driven Compliance Automation," IBM Research GmbH, Tech. Rep. RZ 3662, 2006.
[11] L. T. Ly, S. Rinderle-Ma, D. Knuplesch, and P. Dadam, "Monitoring Business Process Compliance Using Compliance Rule Graphs," in *On the Move Conf. (OTM)*, vol. 7044, 2011, pp. 82–99.
[12] L. T. Ly, S. Rinderle-Ma, K. Göser, and P. Dadam, "On enabling integrated process compliance with semantic constraints in process management systems," *Information Systems Frontiers*, vol. 14, no. 2, pp. 195–219, Apr. 2012.
[13] M. Strembeck and J. Mendling, "Modeling process-related RBAC models with extended UML activity models," *Inf. Softw. Technol.*, vol. 53, pp. 456–483, 2011.

[14] L. T. Ly, S. Rinderle-Ma, and P. Dadam, "Design and Verification of Instantiable Compliance Rule Graphs in Process-Aware Information Systems," in *Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, 2010, pp. 9–23.
[15] A. Birukou, V. D'Andrea, F. Leymann, J. Serafinski, P. Silveira, S. Strauch, and M. Tluczek, "An Integrated Solution for Runtime Compliance Governance in SOA," in *Int. Conf. on Service-Oriented Computing (ICSOC)*, vol. 6470, 2010, pp. 122–136.
[16] A. Awad, E. Pascalau, and M. Weske, "Towards Instant Monitoring of Business Process Compliance," *Entwicklungsmethoden fuer Informationssysteme und deren Anwendung (EMISA) Forum*, vol. 30, no. 2, pp. 10–24, 2010.
[17] D. Knuplesch, M. Reichert, and A. Kumar, "A framework for visually monitoring business process compliance," *Inf. Syst.*, vol. 64, pp. 381–409, 2017.
[18] Y. Liu, S. Muller, and K. Xu, "A static compliance-checking framework for business process models," *IBM Systems Journal*, vol. 46, pp. 335–362, 2007.
[19] F. Daniel and M. Matera, *Mashups - Concepts, Models and Architectures*. Springer, 2014.
[20] J. Yu, B. Benatallah, F. Casati, and F. Daniel, "Understanding Mashup Development," *IEEE Internet Computing*, vol. 12, pp. 44–52, 2008.
[21] W. Kongdenfha, B. Benatallah, J. Vayssière, R. Saint-Paul, and F. Casati, "Rapid development of spreadsheet-based web mashups," in *The Web Conf. (WWW)*, 2009, pp. 851–860.
[22] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn, "Taverna: a tool for building and running workflows of services," *Nucl. Acids Res.*, vol. 34, pp. 732, W729, 2006.
[23] L. Baresi and S. Guinea, "Mashups with mashlight," in *Int. Conf. on Service-Oriented Computing (ICSOC)*, 2010, pp. 711–712.
[24] C. Cabanillas, M. Resinas, and A. Ruiz-Cortés, "Hints on how to face business process compliance," in *JISBD Workshops (PNIS)*, 2010, pp. 26–32.
[25] A. Förster, G. Engels, T. Schattkowsky, and R. V. D. Straeten, "Verification of Business Process Quality Constraints Based on Visual Process Patterns," in *IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE)*, 2007, pp. 197–208.
[26] S. Sadiq, G. Governatori, and K. Namiri, "Modeling Control Objectives for Business Process Compliance," in *Int. Conf. on Business Process Management (BPM)*, 2007, pp. 149–164.
[27] A. Awad, G. Decker, and M. Weske, "Efficient Compliance Checking Using BPMN-Q and Temporal Logic," in *Int. Conf. on Business Process Management (BPM)*, vol. 5240, 2008, pp. 326–341.
[28] O. Türetken, A. Elgammal, W. van den Heuvel, and M. P. Papazoglou, "Enforcing compliance on business processes through the use of patterns," in *EU Conf. on Inf. Syst. (ECIS)*, 2011, p. 5.
[29] S. Goedertier and J. Vanthienen, "Designing Compliant Business Processes with Obligations and Permissions," in *BPM Workshops*, 2006, pp. 5–14.
[30] A. Ghose and G. Koliadis, "Auditing Business Process Compliance," in *Int. Conf. on Service Oriented Computing (ICSOC)*, 2007, pp. 169–180.
[31] I. Weber, G. Governatori, and J. Hoffmann, "Approximate Compliance Checking for Annotated Process Models," in *CAiSE Workshops*, 2008.
[32] M. Montali, F. M. Maggi, F. Chesani, P. Mello, and W. M. P. van der Aalst, "Monitoring business constraints with the event calculus," *ACM Trans. Intell. Syst. Technol.*, vol. 5, no. 1, pp. 17:1–17:30, 2013.
[33] F. M. Maggi, M. Montali, M. Westergaard, and W. M. P. van der Aalst, "Monitoring business constraints with linear temporal logic: An approach based on colored automata," in *Int. Conf. on Business Process Management (BPM)*, vol. 6896, 2011, pp. 132–147.
[34] A. Awad, M. Weidlich, and M. Weske, "Consistency Checking of Compliance Rules," in *Int. Conf. on Business Inf. Syst. (BIS)*, 2010, pp. 106–118.
[35] E. Ramezani, D. Fahland, and W. M. P. van der Aalst, "Where Did I Misbehave? Diagnostic Information in Compliance Checking," in *Int. Conf. on Business Process Management (BPM)*, 2012, pp. 262–278.
[36] J. Stevovic, J. Li, H. R. Motahari-Nezhad, F. Casati, and G. Armellin, "Business process management enabled compliance-aware medical record sharing," *Int. J. Bus. Process Integr. Manag.*, vol. 6, no. 3, pp. 201–223, 2013.
[37] P. Silveira, C. Rodríguez, A. Birukou, F. Casati, F. Daniel, V. D'Andrea, C. Worledge, and Z. Taheri, "Aiding compliance governance in service-based business processes," in *Non-Functional Properties for Service-Oriented Systems: Future Directions*, 2011.
[38] E. Mulo, U. Zdun, and S. Dustdar, "Domain-specific language for event-based compliance monitoring in process-driven SOAs," *Service Oriented Computing and Applications*, vol. 7, pp. 59–73, 2013.
[39] I. American National Standards Institute, "Role-Based Access Control. ANSI INCITS 359-2004," http://csrc.nist.gov/rbac, Last accessed in July 2019 2004.
[40] P. Silveira, C. Rodriguez, F. Casati, F. Daniel, V. D'Andrea, C. Worledge, and Z. Taheri, "On the Design of Compliance Governance Dashboards for Effective Compliance and Audit Management," in *ICSOC 2009 Workshops*, 2010, pp. 208–217.
[41] F. Arbab, "Reo: A Channel-based Coordination Model for Component Composition," *Mathematical. Structures in Comp. Sci.*, vol. 14, no. 3, pp. 329–366, 2004.
[42] Z. E. Akkaoui and E. Zimanyi, "Defining ETL worfklows using BPMN and BPEL," in *Int. Workshop on Data warehousing and OLAP*, 2009, pp. 41–48.
[43] C. Cabanillas, M. Resinas, A. del Río-Ortega, and A. Ruiz-Cortés, "Specification and Automated Design-Time Analysis of the Business Process Human Resource Perspective," *Inf. Syst.*, vol. 52, pp. 55–82, 2015.
[44] C. Cabanillas, D. Knuplesch, M. Resinas, M. Reichert, J. Mendling, and A. Ruiz-Cortés, "RALph: A Graphical Notation for Resource Assignments in Business Processes," in *Int. Conf. on Advanced Inf. Syst. Eng. (CAiSE)*, vol. 9097, 2015, pp. 53–68.
[45] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016.
[46] R. A. Gandhi and S. W. Lee, "Discovering multidimensional correlations among regulatory requirements to understand risk," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 4, pp. 1–37, 2011.
[47] C. Cappiello, F. Daniel, A. Koschmider, M. Matera, and M. Picozzi, "A quality model for mashups," in *Int. Conf. on Web Engineering (ICWE)*, 2011, pp. 137–151.

Dr. Cristina Cabanillas is a post-doctoral researcher at the ISA Research Group of the University of Seville (Spain). She is currently coordinating the CONFLEX project on the integration of context-aware resource management into flexible process-oriented organisations. Her main research interests relate to business process management with a especial focus on their organisational perspective.

Dr. Manuel Resinas is an Associate Professor at the University of Seville (Spain), and a member of the ISA Research Group. His current research lines include analysis and management of service level agreements, business process management, process performance analytics, and cloud-based enterprise systems. Previously, he worked on automated negotiation of service level agreements.

Prof. Dr. Antonio Ruiz-Cortés is a Full Professor and head of the Applied Software Engineering Group at the University of Seville (Spain). His current research focuses on service-oriented computing, business process management, testing and software product lines. He is an associate editor of Springer Computing, recipient of the Most Influential Paper of SPLC (2017) and VAMOS award (2020), and elected member of the Academy of Europe.