universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"Runtime Compliance and Security Monitoring in Process-Aware Information Systems"

verfasst von / submitted by

Patrik König, BSc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Wien, 2017 / Vienna, 2017

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 926
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Wirtschaftsinformatik
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Math. Dr. Stefanie Rinderle-Ma

Abstract. Process Aware Information Systems (PAIS) are Information Systems that are dedicated to distributing and supervising work, according to a strict set of requirements. Their goal is to support the creation of products according to a strictly specified process. But apart from the requirements to achieve that goals often a additional set of external rules (e.g. laws) exist, that have to be adhered to as well. Thus in PAIS the need for monitoring and compliance checking these rules is omnipresent. While monitoring deals with finding out what is going on, compliance checking is about finding out if the rules are violated or not.
Currently these external rules are often built into the processes, which means that it is often hard to understand, what is necessary for creating a result, and what are necessities imposed by external sources. Multiple concepts for compliance monitoring systems exist which separate these aspects. In this thesis common compliance aspects like access control, synchronization of resources and resource monitoring are discussed, followed by a collection of general requirements for compliance monitoring and checking systems. The gathered insights were evaluated, by first creating a prototypical implementation which was then tested and further refined to be as generic and easy to use as possible.

Keywords: Business Process Management Systems, Process-Aware Information Systems, Compliance Monitoring, Business Process Compliance

Zusammenfassung. Prozessgesteuerte Informationssysteme (PAIS) sind Informationssysteme, welche dafür konzipiert sind Aufgaben zu verteilen und deren Ausführung zu überwachen. Dieser Vorgang geschieht hierbei auf Basis von klar definierten Prozessen welche die Anforderungen an das Produkt oder die Leistung widerspiegeln. Die Leistungserstellung wird hierbei nicht nur von internen Faktoren beeinflusst, sondern obligt auch externen Einflüssen (wie etwa Gesetze). Diese Regularien führen zu einer Fülle an Anforderungen, welche während der Leistungserstellung berücksichtigt werden müssen. Dies führt zum Bedarf einer umfassenden Lösung um die Compliance zu überwachen und zu kontrollieren. Die Überwachung beschreibt hierbei die fortlaufende Beobachtung der laufenden Prozesse und die Kontrolle beinhaltet deren Validierung anhand definierter Regeln.
Aktuell sind externe Regeln meist direkt in den Prozessen inkludiert, somit ist es meist schwer nachzuvollziehen, ob und welche Schritte welchen Regularien unterliegen. Hierbei existieren bereits mehrere Konzepte welche sich dem Thema Compliance Überwachung widmen und besagte Regeln vom Prozessmodell separieren und neu strukturieren. In dieser Masterarbeit werden folglich relevante Themen des Compliance-Monituring diskutiert und die Anforderungen an ein Compliance-Monitoring und Checking-System evaluiert. Basierend auf den daraus erhaltenen Einblicke wurde eine prototypische Implementierung erstellt. Diese Implementierung wurde anhand der Anforderungen getestet und iterativ weiterentwickelt um so generisch und benutzerfreundlich wie möglich zu sein.

Acknowledgement This master thesis has accompanied me for a long time now, from a first draft, to the first setback, right up to this final work. It has given me the chance to learn and grow with its challenges. Some of these insights are printed on the following pages, but most of them lie in between them. I would like to thank my parents and especially my girlfriend Rebecca for their support during my studies and throughout the last years.
I also would like to thank my supervisor Stefanie Rinderle-Ma for giving me the chance to be a part of the research group. Therefore I would like to thank every member of the research group "Workflow Systems and Technologies" for sharing their knowledge and passion on the topic with me. Especially I want to thank Jürgen Mangler for being a mentor and helping me accomplish this work. Additionally I want to thank Florian Stertz for his help and the proof-reading of this thesis.
Last but not least I would like thank my friends for their help and support.

Table of Contents

1 Introduction

1.1 Motivation

Processes have to comply to internal as well as external regulations. Regulations (e.g. laws) are becoming more and more complex. Consequentially the field of process compliance became an emerging topic over the last years. This has also been stated in [1] and is reflected by the vast amount of publications discussing this topic.

This thesis addresses run-time checking and monitoring of process compliance. As more exhaustingly reflected in Sect. 1.4, there are several approaches aiming at the evaluation of different process model life-cycle stages (i.e. design time, run-time, ex-post). Within this scope different problems arise. One of the major problems is, that not all information (e.g. price of parts not available at design time, availability of resources at certain stages of runtime, ...) is present at any time of the evaluation. Therefore gathering the needed information is a challenge, when for example, checking if a certain machine does not exceed the output of allowed CO2 values.

One framework which is considered as a source for compliance monitoring and checking functionalities is the CMFF (Compliance Monitoring Functionalities Framework). Hereby there is currently no system which fully complies to more than six out of nine aspects specified in CMFF [1].

As a starting point the compliance concepts described in the SPRINT (Security in PRocess-Aware INformation SysTems) approach [2] will be considered as a general construct for the definition of compliance rules. Contrary to CMFF yields to define general requirements on compliance monitoring and checking, SPRINT proposes a language concept to define policies for PAIS. This will be complemented by a general analysis of compliance aspects.

The aim of this master thesis is therefore to collect the requirements for a compliance monitoring and checking system and to implement a prototype. Furthermore aspects and problems of the implementation of a compliance monitoring and checking system are discussed in detail. In this thesis a lot of feedback through discussions in the CRISP project (funded by FWF project number ICT Call 2015 / ICT15-072) was elaborated.

1.2 The Process Life-Cycle

The validation of process compliance can be done within different stages of the process life-cycle. In general three basic stages can be identified [1,4]:

- Design-time: the process is created.
- Run-time: a instance of the process is created and executed.
- Ex-post: after the execution of the instance.

Compliance evaluation and monitoring approaches may be distinguished based on the stage of the life-cycle they operate on.

Design-time Compliance Evaluation. One of the approaches to evaluate process compliance is the verification of the process model during design-time. This means even in the design phase of a process, possible compliance violations can be checked. There are plenty of approaches, ranging from static evaluation against predefined rules up to checking the possible result-space of a process as well as an evaluation against certain qualitative constraints for the model[4]. Therefore an evaluation can be done even before the process is executed even once, which makes this approach an excellent way to check and monitor maturity of a yet not implemented or executed process. This in general should also reduce the risk of possible faults during execution.

The major drawback is that a whole class of rules can not be checked: rules that deal the restriction of data values, as these values are not available during design time, but are only available at runtime.

Further model checking for processes (based on compliance rules) is np-hard. While it is possible to check the validity of processes for an integer variable that might occur in decisions in the process (check all possible integer values), it becomes near practically impossible for strings. Even though there are approaches for a dynamic evaluation, these approaches tend to reduce the expected possibilities in some way, causing a trade-off between flexibility and control [4].

Run-time Compliance Evaluation. In run-time evaluation, live event streams or logs are monitored and evaluated against different rules or processing logics to determine potential violations of predefined policies or unexpected behaviour. Similar to design-time compliance evaluation there is a huge range of different scopes for what and how to validate process execution. But in contrast to design-time evaluation, run-time evaluation is focused on instances rather than the

process model itself. Compared to design-time evaluation, this instance view in general reduces the risk of potential "state-explosions" [4] caused by the possible process permutations.

An additional aspect which can be introduced with run-time compliance checking is the consideration of inter-instance constraints [5]. Those are constraints which span across multiple process instances, considering (compared to design-time evaluation) new important scopes such as resource allocation for multiple parallel instances.

Shortcomings in run-time process compliance evaluation are at most, that many approaches do not include capabilities to consider or even predict future process behaviour. As discussed in [4] such concepts predict and therefore avoid possible violations. This lack of prediction capabilities primarily arises from the time-critical aspect of run-time evaluation, therefore multiple conditions need to be checked in real-time, thus complex and particularly time consuming processing is being avoided. Hereby it is assumed that the considered process step has to wait for the evaluation to complete. Additionally during evaluation at run-time, it may be the case that not every bit of information is available at the moment or does not even exist.

Ex-post Compliance Evaluation. Ex-post or a posteriori compliance evaluation is in most cases used in combination with the term process mining. Hereby it is also referring to methods or methodologies extrapolating data from one or more sources, cf. [6]. This greatly makes sense for process auditing as well as statistical analysis. Besides the possibility that certain information has not been logged, this stage of the process life-cycle has the most analytical capabilities. This capabilities arise as the full process instance trace is available, including full information of executed instances as well as possible meta-data.

Considering process compliance evaluation, the major drawback of a validation within this stage of the process life-cycle is the lack of possibility to immediately respond to non-compliance. This is the case because the analysis is, by definition, after the process execution has finished. Therefore the violation, or more general non-compliance, already took place and probably altered the process outcome.

On the other hand in most ex-post analysis scenarios, only a limited set of data is available for analysis. The process mining manifesto [6] for example specifies five different levels of event log data maturity based on the granularity of the event logs and semantics within them. Based on the different challenges of

data and process mining, dealing with various data structures and quality, the analysis methods use a generic approach mostly based on process structure or behaviour cf. [7,8].

1.3 Research Questions and Contributions

For the compliance monitoring and checking the following research questions are discussed:

- What views on business process compliance exist and how can they be expressed?
- What are the requirements to implement a compliance monitoring and checking approach?
- Which modular design of a prototypical compliance monitoring and checking system allows for maximum flexibility regarding the implementation of possible future requirements?

Based on these research questions, this thesis contributes the following three topics to state of the art in Compliance Management:

1. This thesis elaborates on the general requirements of a Compliance Management system for PAIS and the common challenges of the implementation of such an system. This thesis also discusses a broader scope of security aspects, as an extension to [9].
2. This thesis outlines how to implement such a system based on the generic requirements. Two concrete approaches are discussed to evaluate the validity of the approach.
3. This thesis gives a definitive guideline for implementing relevant aspects of a Compliance Management system for PAIS according to the state of the art, based on the requirements and the prototypical implementations.

The final outcome is a data driven, highly flexible Compliance Management system. This is characterized by a lose coupling to other information system components and its basically generic data structure. Furthermore compared to other approaches no process repository is required and the system solely operates on the received events.

The resulting Compliance Management system includes two iterations which are both provided under: `http://gruppe.wst.univie.ac.at/research/index.php?m=D&t=service95&c=show`.

1.4 Scope

There is a variety of aspects of process compliance, therefore some basic assumptions are made. These assumptions are based on definitions given in related work.

The scope of this master thesis and furthermore the implementation in terms of the process life-cycle is focused on **run-time** compliance checking. Which means the compliance monitoring and checking will take place during execution of a given process on a given information system.

The terms conformance checking and compliance monitoring are related but can not be used synonymously. This work relates to compliance monitoring, in terms of information systems. Therefore it encompasses the computer-supported real-time evaluation of the behaviour of IT-systems based on predefined compliance rules. Other types of compliance monitoring exist in other domains, e.g. in accounting or regarding the ethical behaviour of management staff. Conformance checking on the other hand means to analyse if or how well a process instance complies to a given process model. This is also generally applied afterwards (ex-post), on any output files, like log files [1].

Secondly, it is not intended to create or define an information system which includes all the compliance concepts itself. The desired Compliance Management system should generic enough to be loosely coupled with arbitrary existing process engine. For the implementation, the parts of the monitoring system which interact with the PAIS tend to be highly adapted to information the PAIS provides. This thesis relies the PAIS providing real-time recurring events for all aspects of process execution. Thus one of the outcomes of this thesis will be an analysis,of the minimal required events that are necessary to realiye a compliance monitoring and checking system.

In Sect. 2.2 different aspects concerning compliance are discussed and defined. Apart from this, it is not intended to discuss how business rules or security policies for an information system can be derived. For this aspects the SPRINT compliance language [2] in section: “Security Policy Acquisition” is used. In the referenced literature a concrete methodology for compliance rule acquisition is proposed.

The compliance concept will be primarily based on the SPRINT approach which defines structures to define such compliance rules [2]. The main reason SPRINT (“Security in PRocess-Aware INformation SysTems”) is considered as starting point is because it already includes different aspects for structural as well as organisational constraints. This and the possibility of a fine granularity

for the different compliance-rules offer a holistic approach regarding compliance as well as security aspects [2].

Furthermore general aspects of compliance and justification of this decision are reflected in Sect. 2.2.

The concepts of SPRINT itself are discussed in Sect. 2.4.

1.5 Methodology

For the implementation of the Compliance Management system, first a literature research was conducted to define the basic scope of the implementation. Based on the related work, common requirements of the system were derived. The system and information structure was analysed to determine the basic information flow. A first prototype was implemented and relevant structural problems were identified. Based on the lessons learned, further requirements and properties of the desired architecture were established. According to these insights a second prototype was implemented. The resulting prototype is hereby referred to as COMS (COmpliance Management System).

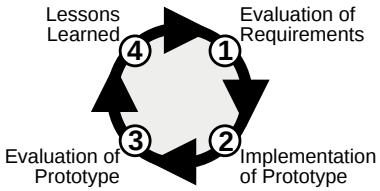


Fig. 1: Methodology

Fig. 1 shows a generic development cycle, which describes the applied methodology [3]. The first part of one iteration covers the evaluation of the requirements for the system. The next steps refer to the implementation and the evaluation of a prototype. Lastly, there is also a lessons learned step, which is taken into account for the next iteration. The first step, Evaluation of Requirements, usually takes most of the time in the first iteration.

1.6 Structure of Thesis

This thesis is structured into four main parts. The first part describes the theoretical basis of the topic. The second part addresses the underlying information system architecture. The next part outlines the incremental implementation approach. Finally we discuss the results and future work.

In detail, the following overarching aspects are very important for this work: within the definition of PAIS, a further distinction to the relevant characteristics of a modern PAIS is made. This definition includes the key requirements for the Workflow System on which the compliance monitoring and checking system later is applied. For the discussion of process compliance in general, the scope of different approaches describing process checking and compliance is considered. Subsequent to the analysis of aspects and approaches on process compliance and security, the CMFF is described and reflected. The CMFF introduces a generic view on the topic and will be considered as a main source, defining the basic functionalities of the implementation. After the definition of desired compliance monitoring and checking functionalities, the choice of SPRINT as a basis for the compliance concept is vindicated and its basic concepts are discussed. Additionally, viewpoints and concepts of process instance life-cycle models are analysed, as they turned out to be a mayor aspect to be considered within the implementation of COMS.

The chapter "Requirements and Architecture" then focusses on the description and analysis of the incorporated information systems, beginning with the description of the main characteristics of the CPEE as underlying Process Execution Engine. This is then followed by an overall architectural of description of the compound information system structure. This is continued by an analysis of the relevant message and information flow.

By having evaluated the general requirements and a rule concept as starting point, the first implementation was carried out. During the finalization of the first implementation insights of structural problems arose. This insights are reflected in the subsequent section. Based on those learnings a second implementation was done. As the second prototype offers the possibility to represent the scoped requirements, a broader evaluation against the scoped requirements has been carried out.

1.7 Definition of Terms

BPAF Business Process Analytics Format, is an XML language which defines a formal process model based on a unified state machine. Including both activity and process states [10].

BPEL Business Process Execution Language, is an XML language which is used for the definition and execution of business processes using web services. It enables the definition of executable business processes by using common web technologies [11].

BPMN Business Process Model and Notation, is a graphical modelling language for the specification of business processes.

CMFF Compliance Monitoring Functionalities Framework, a published framework which specifies current relevant requirements and aspects on compliance monitoring.

Compliance Management In this thesis, the term Compliance Management describes the process of monitoring and checking of business processes. This includes the definition and enactment of compliance rules.

COMS COmpliance Management System, is the name of the software artefact resulting from this thesis.

CPEE Cloud Process Execution Engine, is a highly flexible, process execution engine.

DSL Domain Specific Language, defines a programming language for a specific purpose .

event An event, in terms of distributed information systems, represents a chunk of information. In this context, an event is sent from one system to another, notifying about an action that took place in the source system.

JSON JavaScript Object Notation, is a text based data transfer format.

LTL Linear Temporal Logic, is a logic based language which enables the description of a temporal order of events.

MXML Mining eXtensible Markup Language, is an XML language defined to promote a standard description for process execution logs.

PAIS Process Aware Information System, is a workflow system that is integrated with other tools that are required in a certain domain. For example Customer Relationship Management (CRM) or Manufacturing Resource Planning (MRP). A broader definition is given in section 2.1.

Process Engine A Process Engine interprets a process as a program, that consists of sequential and parallel tasks. This program is executed. Each task deals with assigning a certain unit of work to a variety of resources (humans, machines, computing nodes). Tasks require input data and yield output data. The output data is saved and is used as input for other subsequential tasks.

REST REpresentational State Transfer, describes a software paradigm for Web Services, in which the web services are stateless and based on common internet technologies.

SPRINT Security in PRocess-Aware INformation SysTems, is a policy definition concept which defines compliance and security rules for PAIS.

WebSocket The WebSocket protocol is a bi-directional network protocol for client - server communication, which is commonly used in web applications [12].

Workflow System A Workflow System is an Information System that deals with the distribution of work to a variety of resources (humans, machines, computing nodes) in the context of running processes.

Worklist An external piece of software that is triggered by a Process Engine and is responsible of balancing the work (that is represented by a single task) between multiple eligible human resources. It is akin to a load balancer for multiple computing nodes.

XES eXtensible Event Stream, is the successor of MXML. XES aims to provide a unified and extensible XML language for capturing process execution logs [13].

XML Extensible Markup Language, is a hierarchical structured markup language.

YAML YAML Ain't Markup Language, is a data serialization language designed to be human-friendly [14].

2 Related Work

2.1 PAIS

The term "Process-Aware Information System" will be used frequently in this thesis. Therefore apart of the common definition of the term, it is necessary to define the scope and the expected functionalities of such a system.

In a broad sense an information system is process-aware if its execution structure is based on defined process steps, commonly referred to as activities. Or as defined in "Process-aware Information Systems, Bridging People and Software Through Process Technology" [15]:

> *".. a software system that manages and executes operational processes involving people, applications, and/or information sources on the basis of process models.."*

[15] also includes the following definition:

> *"An information system is called process-aware if it supports process enactment by scheduling the activities according to the specified rules of the respective process type"*

Types of PAIS. The general definition, which was introduced at the beginning of this section, would also include information systems which express business processes implicitly within their source-code.

To such information systems which incorporate a static definition of the process models, it is referred to as "specific process support system", cf. [15]. This type of information systems defines a rather static process model as a change in a process would also involve an adaption of the source-code.

Contrary to a "specific process support system", the system under study in all concerns is a information system referred to as "generic system". This term specifies a Workflow System which abstracts the process execution from the process model, meaning that such a generic system does not incorporate information about the structure of processes by default [15]. This aspect adapts to most of the so-called "Process Engine".

Business Processes typically are available in a standardized format like BPEL or BPMN. In order to connect real-live services to the activities in these descriptions they have to adhere to certain interfaces. Thus business processes are often

not easily realized with these systems. In most cases either the processes have be adapted to real-live services the involved services have to be adapted to fit. Furthermore not all "Process Engines" support the same formats, or even rely on the same interfaces for activities. This often results in an environment that is highly tailored towards certain technology choices, which makes the system ineligible to change.

Contrary to such a system, the system considered in this thesis is highly adaptable to process changes and uses standard, state-of-the-art technologies for execution and service communication.

Adaptive PAIS. In general within a PAIS, activities are executed based on process models, therefore users are not eligible to change the execution-order itself. The thread of control in general is only altered by the data which is provided to the information system as output from invoked activities.

To enhance flexibility and make systems more eligible to change, there are different approaches for a more flexible process definition [16] as well as languages and concepts to alter running processes in a semantically correct manner [17,18]. Such a flexible/adaptive system it is generally referred to as **adaptive** PAIS [9].

Such an adaptive PAIS also sets additional requirements for the compliance monitoring and checking approach. The Compliance Management system must be able to deal with changes made during process execution.

This requirement therefore generally excludes approaches for compliance evaluation done on a structural level, which only utilize model checking approaches [19,20,21] operating on reference-processes or process repositories. As for updates in the process model, they are mostly not supported or at least have to be done on the process as well as the compliance rule level [20].

2.2 Compliance

Within this chapter, the scope of process compliance for this thesis is defined and different views are discussed. It aims to describe the basic characteristics a suitable compliance language should consider. To establish this general view on compliance, basic concepts and viewpoints on the topic are derived from literature.

Definition. A compliance rule language or concept in terms of business processes, states what aspects are concerns of compliance and how they can be described. Therefore the compliance rule language realizes a specific syntax so that it is able to describe the desired aspects (semantic) in a structured manner.

Scopes of Compliance Languages. As there are different views on how to describe process compliance, there is a corresponding number of different approaches [4,22,7,20]. Based on the concept, the compliance languages can vary in breadth and depth of the described compliance aspects.

- For **breadth** the Modelling Based Requirements of CMFF [1] are conducted. The CMFF defines time, data and resource constraints which can be seen as the basic aspects a compliance language does refer to.
- The **depth** is an aspects of maturity and coverage of the language within each of the three constraint domains. Some compliance checking concepts are highly specialized in one particular domain. E.g. [20] mostly discusses qualitative time constraints and doesn't the the resource domain into account. [23] and [24] on the other hand mainly target resource aspects in terms of organizational models.

Despite the scope, there are as well different views how compliance can be described for business processes. This focuses primarily on how the rules are defined in respect to the process model and what the context of the respective rule is.

Business Rules. A Business Rule relates to business guidelines or goals defined by an organization itself, or policies imposed by external sources such as lawmakers, e.g. to enforce quality standards or consumer regulations [25].

Modelling Concepts. There are different ways how a business process and the business rules can be expressed. There are two main approaches are [9]:

- The imperative approach, which is the traditional most common used way, defines a specific sequence in which activities are executed. Such an imperative model is shown Fig. 2.
- The declarative approach [16] does not specify the control flow of the model but defines how the different activities are related to each other. A declarative model (ConDec) is shown in Fig. 3.

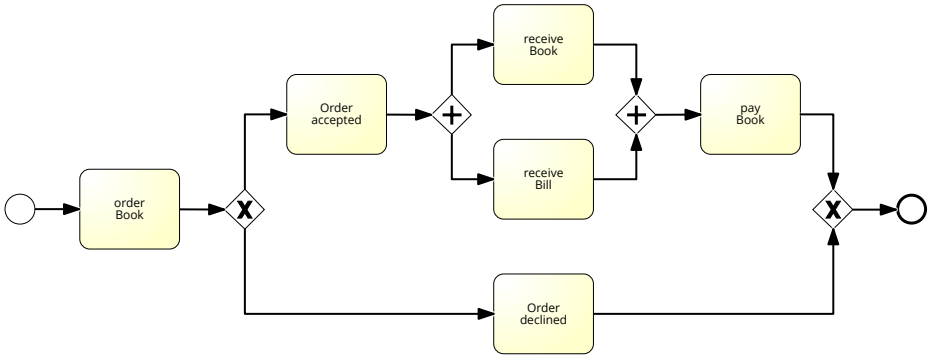


Fig. 2: Book order process, defined in an imperative Model (BPMN)

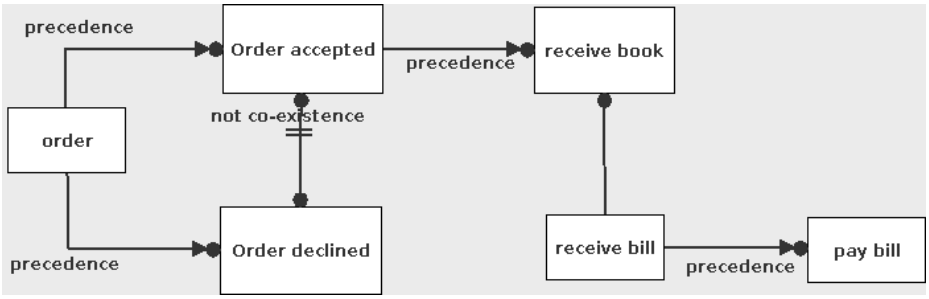


Fig. 3: Book order process, defined in a declarative Model cf. [16]

In general the imperative approach is the most straight forward one but can be very exhausting in dynamic, fast changing business environments. For example, as rules are expressed within the model, the process needs to be adapted

with every change in the rule-set [9]. Contrary to that, declarative languages loose a lot of visual expressiveness, as parts of the information included in the control flow is no longer included in the visual notation of the process model. Also declarative languages are very disruptive to the widely accustomed imperative modelling approach, as the paradigm changes from a mostly visual to a more semantic approach, including temporal logic and highly specialised annotations as in DECLARE [26] or ConDec [16].

Other literature [27] summarizes this differences: an imperative model has a higher expressive capability for sequential process information and a declarative model establishes more circumstantial information for the process.

The Process Model and Rule Model. Apart from the process model itself, business rules can either be incorporated (internal) into the process model or expressed external with a separate rule model. As shown in Fig. 4, this leads to multiple possible combinations of the two modelling approaches.

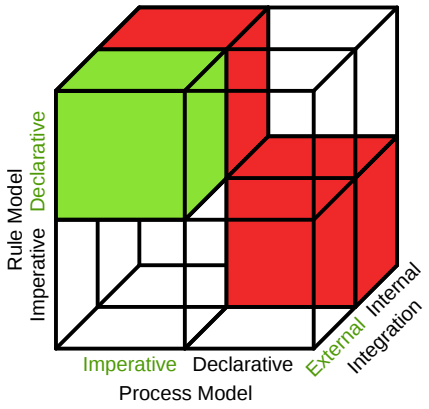


Fig. 4: Combinations of process and rule model concepts

Internal in this case means that the business rules and the process model are represented in one single model. Hereby two combinations which are not feasible are marked in red within Fig. 4. This two combinations are pointless as they would integrate the two different modelling approach into one single model. This makes for a technology mix with is hard to link together, as the process engine has to be both: a rule engine and a imperative language interpreter.

– Process imperative, rule imperative

- Internal:
  The business rules are integrated into the business process. In this case it is hard to separate which parts of the process defines the rules and which part represents the general process flow.
- External:
  The business rules and the business process are modelled separately. During enactment of the process model, the process model and the business model would need to be synchronized and checked against each other. Expressing rules as imperative code pieces, requires that the imperative process model structure has to be enriched with synchronization information, which essentially doubles the effort for this approach.

– Process declarative, rule imperative
  - Internal:
    Not feasible.
  - External:
    Problem: a declarative model translates in to many possible sequences of tasks, not only one, like an imperative model. This means to build valid imperative rule models, most of the elements in the declarative process model needs to be defined in the imperative one.

– Process declarative, rule declarative
  - Internal:
    The business process description as well as the business rules are all defined in the declarative model. This is efficient and performant, but the result is hard to understand for humans.
  - External:
    This would consist of two declarative models, whereby one describes the process and the other model defines the restrictions. This is pointless as a single rule engine can handle both.

– Process imperative, rule declarative
  - Internal:
    Not feasible.
  - External:
    This includes a straight forward imperative business process, for which all business rules are expressed in a declarative manner. We think this combines the best of two worlds, as all rules are grouped by process instances (a minimal number of rules allows for humans to easily understand what is going on).

The most desirable combination is the one which is marked as green in Fig. 4. This combination is an imperative process model, with an external declarative rule model. The combination is preferred as the best of both worlds can be used. Hereby the business process solely describes the process flow which produces added value, and the rule model defines the restrictions for this process model by defining relevant relations.

Security Policies. Compared to business rules, security policies do not directly aim to enforce corporate policies and regulations but are intended to protect the system against intentionally or unwanted fraudulent change.

For aspects of information security the taxonomy most commonly used is the CIA-triad (Confidentiality, Integrity, Availability). Shown in Tab. 1 and described in [9] a syllogism to this principles in context of PAIS is derived.

Table 1: Security policy taxonomy in context of PAIS [9]

Security Goal	Context of PAIS
Confidentiality	Information processed and available in a PAIS should only be accessible to authorized users.
Integrity	Information can only be altered by process stakeholders which are explicitly authorized and only within a designated context.
Availability	May refer to the PAIS itself or the services it invokes and data it consumes.

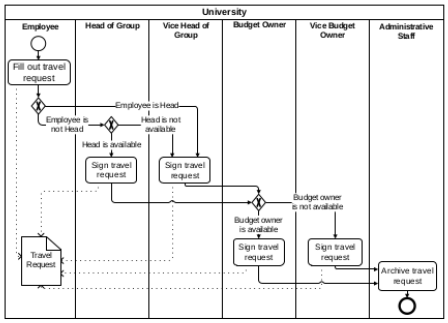
In Information security literature the adequacy of the CIA-triad has been questioned, as it is very generic and possibly does not catch the complexity of current information systems [28]. Therefore an extended schema is proposed in [29] which adds the aspects of accountability, auditability, authenticity/trustworthiness, non-repudiation, privacy to the CIA-triad taxonomy, in order to set those additional aspects [29] listed in Tab. 2 and set in a narrow context to PAIS.

Expression of Security Policies. For the definition of Security policies in general four ways of how they are linked to the model, have been identified [9]. Examples for this representations are shown in Fig. 5.

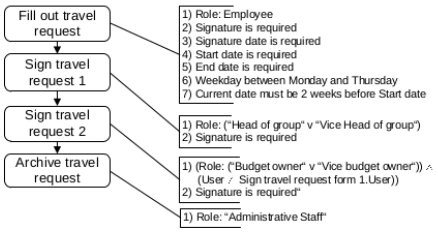
- **Inherent**, means that security policies are included in the control- and data flow of the process and expressed implicitly within this concept.
- **Attached** security policies are defined explicitly for a certain process activity on which they apply, so they can be seen as an attribute of the activity
- **Separate** security policies are defined separately, redundant to the control- and data flow and are mapped to the process e.g. at run-time.
- **Task-based** representation as defined in [30], enriches its control model with mutability and continuity aspects. To achieve this the access control relation between Object and Subject is interfered by a Rights context holding additional Authorization, Obligation and Condition information and is continuously checked.

This aspect, of how the security policies are expressed, may strongly correlate to the way Business rules are defined. This especially applies for the declarative approach, in which by definition rules are not expressed directly within the control-flow. Therefore an inherent definition of security policies would not be reasonable.

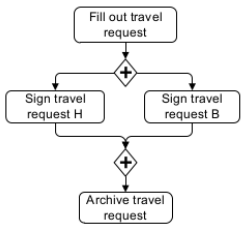
Within the inherent, attached and task-based representation policies are directly bound to a specific process model utilize different granularity. Due to this in the separate representation a mapping function is needed to map the loosely defined policies to the corresponding tasks [2].



(a) Inherent Representation



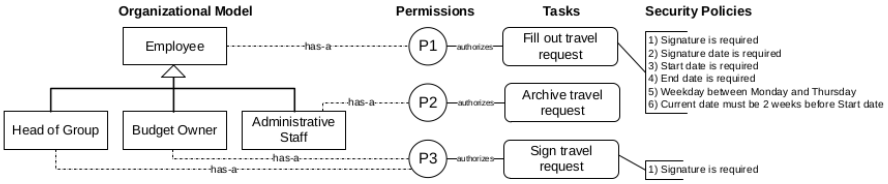
(b) Attached Representation



Security Policies

- An employee has to sign the request before a head or budget owner can sign the request.
- An employee has to file a request two weeks before the traveling starts.
- A travel req. can only be filed between Mondays and Thursdays (for accounting reasons).
- A travel request applies only to employees of a specific faculty (e.g., A,B,C) at University. (Other faculties at University may have different procedures.)
- A travel req. has to be signed by a head of group and a budget owner.
- A travel req. has to be approved by two different persons.
- If an employee is the head of group, then a vice head has to authorize the travel request.
- In case a head of group is not available a vice head is authorized to sign the request.
- If a budget owner (employee might be budget owner) is not available, then a vice budget owner is allowed to sign the request.

(c) Separated Representation



(d) Task-based Representation

Fig. 5: Different Representations as shown in [2]

Table 2: Additional aspects of information security [29] in context of PAIS

Security Goal	Definition	Context of PAIS
Accountability	The ability of a system to hold users responsible for their actions (e.g. misuse of information)	It is replicable in a PAIS which user executed a certain task
Auditability	An ability of a system to conduct persistent, non-bypassable monitoring of all actions performed by humans or machines within the system	This can be related to the existence of a proper monitoring system which evaluates the behaviour of the PAIS, as well as the capability of a PAIS to expose its state.
Authenticity/ Trustworthiness	An ability of a system to verify the identity and establish trust with a third party and the information it provides	In a PAIS it is possible that external services are invoked for the enactment of tasks. The key concerns of the PAIS or furthermore the monitoring system are to verify the authenticity and integrity of those external services.
Non-repudiation	The ability of a system to prove (with legal validity) occurrence/non-occurrence of an event or participation/non-participation of a party in an event	In terms of PAIS this can be interpreted as storing all historical run-time information accumulating during the execution of an instance in a way, so that it can not be tampered with.
Privacy	A system should obey privacy legislation and it should enable, where feasible, individuals to control their personal information (user-involvement)	The legally compliant processing of sensible private information in PAIS should be considered at design time and also monitored during execution.

Conclusion. For the purposes of this work, a compliance approach is encompassed with a scope that is broad enough to be able to define rules for all relevant security aspects as well as business rules. It also has to be flexible enough to be adapted to an existing workflow system.

It is preferable that in the compliance concept, business rules as well as security policies are expressed separately from the business process model. This increases the manageability and adaptability of the compliance language [9] as the different rules are not scattered across the process models. Apart from this its context must be applicable to run-time compliance checking, approaches aiming solely on ex-post process evaluation, cf. [7,8] are therefore not considered.

The adaption of processes and process instances, as described in 2.1, should be supported by the language concept. This at least means that altering of the control flow should be possible as long as it does not actively violate a given compliance rule.

Even if in this chapter business rules and security aspects are disjoined, there are many cases where they serve the same goal based on different views. For example the guideline: "a loan above € 50.000;- can only be approved by the bank manager". This represents a business rule as it describes how certain tasks within a company have to be carried out. On the other hand security aspects would be that only employees with the role manager are capable of manipulating this request. Therefore in this case the business rule operates on a control flow level and the security policies aim to satisfy access control restrictions which can be derived from the business rule.

2.3 Requirements According to the CMFF

In addition to SPRINT as a starting point which defines foundational concepts for the structure, a generic approach to evaluate the completeness of the considered functionality is desirable. The purpose of this section is to evaluate basic requirements and consider them during the design phase. As a baseline we use the Compliance Management functionalities framework[1] (CMFF). The CMFF offers a platform independent view on compliance monitoring functionalities. Additionally the scope of CMFF is not based on concrete concepts, referring to particular scenarios but it rather describes requirements on functionality in a structured and systematic way. This is exemplified by the research questions asked:

> *"Research Question 2 : What are functionalities that are essential for compliance monitoring approaches in business processes?"* [1]

This research question states a fundamental question in this research-area also defining the broad scope of CMFF context.
The other relevant research question is:

> *"Research Question 3 : How can we demonstrate the appropriateness of the identified compliance monitoring functionalities?"* [1]

As the result of research Question 3, Ly et. al came up with a qualitative scale to compare different compliance frameworks.

The CMFF will be used as a library which defines basic aspects and requirements for the system implemented in this thesis.

In the following four sections the core concepts of [1] are outlined, those will be later on be used to evaluate the feasibility of the implementation presented in this thesis.

Compliance Scopes. CMFF [1] defines three main aspects of compliance monitoring functionalities in which specific aspects are aggregated.

1. **Modelling requirements**, focus on the compliance-language itself and its possibilities to specify compliance concepts. Furthermore these concepts should not only aim for aspects of the control flow but also on high-level perspectives.

2. **Execution requirements**, are more focused on the activity life-cycle. In this context, these are the capabilities to process the event stream and gather execution-based information which is attached to the events. In an extend this also applies to the capability to process domain-specific information. This generally can be seen as the capability to extrapolate, manage and process relevant data out of given event-streams.
3. **User requirements**, refers to the ability of a monitoring system to give structured and useful feedback to end users. This feedback, depending on the time it is given and its granularity, can range from generic errors to pro-active recommendations. Stating that an end user should not only know the error but furthermore know what caused it and even what should have been done instead to avoid this violation.

The concrete functional requirements are directly adopted from [1] and will be later referred to, in total CMFF lists ten requirements.

Modelling Based Requirements

CMF1: Constraints Referring to Time. One of the most common family of constraints are dealing with aspects of time. Time in this domain can be expressed in two different contexts, namely *qualitative* and *quantitative* time.[1]

- Qualitative time refers to the temporal sequence of activities, where time is measured as a relative relation between them. Such a constraint in it simplest form could be "A must be executed before B".
- Quantitative time constraint are referring to the metric measurement of time. This could also relate to multiple activities, defining a time frame in which an activity needs to be done, based on the completion of another one. But the constraints in this case always include a metric measure.

CMF2: Constraints Referring to Data. Such constraints in a broader sense are defined to monitor the state of the data produced and manipulated by the information system. The context for these constraints is in most cases very similar, as they describe possible and desired states of a process artefact. But the characteristics, in terms of semantic power can differ. Hereby [1] distinguish between two types of conditions, *unary* and *extended*.

- **Unary conditions** describe a comparison of a single artefact against a given value. For example the loan of a given house should not be above € 40.000;-

Table 3: Summarization and Overview of CMFF [1]

ID	Name	Context
CMF1	Constraints referring to time	Qualitative and quantitative time
CMF2	Constraints referring to data	Unary and extended conditions, activity and case data
CMF3	Constraints referring to resources	Unary and extended conditions
CMF4	Supporting non-atomic activities	Consider different activity states
CMF5	Supporting activity life-cycles	Consider a ordered sequence of activity states
CMF6	Supporting multiple instances constraints	Constraints considering multiple instances
CMF7	Ability to reactively detect and manage compliance violations	Detection, feedback and recovery of and from violations
CMF8	Ability to pro-actively detect and manage violations	Prediction of potential violations
CMF9	Ability to explain the root cause of a violation	Trace which actions caused the violation
CMF10	Ability to quantify the degree of compliance	Metric expression of compliance

– **Extended conditions** compare the state and value of multiple artefacts with different possible measures.

Another aspect for data constraints can be the scope in which they are defined. As data can rely on a specific activity or the whole instance, CMFF [1] differs between **activity data** and **case data**. Hereby case data refers to the whole instance scope and activity data only considers the artefact within the scope of a single activity.

CMF3: Constraints Referring to Resources. Resource based constraints are compliance rules referring to organizational resources and bind these re-

sources to artefacts or activities. In terms of compliance checking a resource can be seen as a special type of data because the attributes which describe the resource are data elements. As of the similar type, the resource constraints scope can also have an unary or extended character analogue to the data constraints. For example a loan request greater than € 40.000;- must be confirmed by the head of the bank. Extended conditions are related to more context based rules as e.g. at the requirement of two signatures, the signature B cannot be made by the same person which did signature A.

Execution Based Requirements.

CMF4: Supporting Non-Atomic Activities. When executing a single activity, multiple states need to be passed until it finishes. Based on this behaviour the compliance monitoring system should also be able to process different activity states. Activities can be seen as non-atomic as long as they have at least two states, which in this case respectively should be *start* and *complete*. Besides this two rudimentary activities states, activities can have a vastly deeper granularity of states. To give a basic example, the time which is consumed by an activity can only be measured correctly if at least a start and an end state are available.

CMF5: Supporting Activity Life-Cycles. As an activity has more than one state, the order of transitions of its states can be considered within activity life-cycle constraints. This life-cycle describes the possible state transitions of a non-atomic activity. To continue the previous example, an activity can only be completed successfully if it has started before.

CMF6: Supporting Multiple Instances Constraints. Multiple instance constraint extend the scope of a constraint from its own single instance to the state of all accessible instances. For multi-instance constraints, all aspects defined under Modelling Based Requirements are applicable but the scope of the included aspects change [1].

This rule can furthermore be related to all running (parallel) or all available instances including finished ones. Other relevant aspects are that the information which needs to be obtained could be included in sibling instances (instances descending from the same process model) or instances from other model-types. This reflects an important distinction as the meaning of an activity (i.e. its input/ouput data) depend on the context of the instance.

Another aspect of multiple-instance constraints is that there are several scenarios where mutliple similar instances are spawned by one process and all spawned child instances must finish before a given process can resume. An example for this could be an order process where multiple goods are ordered, the orders are processed separately and the whole order process can not be completed as long as every single good is finally processed [1].

User Based Requirements.

CMF7: Ability to Reactively Detect and Manage Compliance Violations. Besides detection of possible violations, a monitoring system should give feedback to users what exactly caused the violation and give recommendations how to compensate them [1].

Within the CMFF three different aspects of user feedback are defined:

- Detection, is the ability to detect what caused the violation.
- Feedback, aims to be able to provide feedback about what exactly happened, why the violation accrued and what the impact of this violation is.
- Continuous monitoring, describes the ability of a technical Compliance Management framework to continue monitoring after a given violation. This problem could occur for logic based approaches or when simple automata are used [31].
- Recovery and compensation, is the competence to deal with violations, this describes the ability to adapt or enable different rules as a countermeasure for a given violation.

CMF8: Ability to Pro-Actively Detect and Manage Violations. Pro-active detection takes place even before a violation is caused. This especially is of interest because it can avoid the, in most cases costly, compensation of non-compliance. [1] As for the definition "pro-active detection" is to detect possible or unavoidable **future** violation based on current states [1].

Pro-active detection can be employed with multiple strategies, two of those are:

- Structural prediction, could be done based on a given reference process model to detect future violations based on current state and the entered path.
- Statistical prediction, could be the prediction of possible events based on the statistic analysis of given data. For example on a Monday the error-rate for a given task and a given employee is higher as the rest of the week.

Besides the prediction itself, the steps that are taken to communicate or avoid possible violation are also important aspects [1].

CMF9: Ability to Explain the Root Cause of a Violation. Explaining the root cause means that beyond providing a counterexample, the system is able to provide a description of what exactly caused the violation. This means the concrete input or chain of inputs are determined which led to the violation, derived from the actual event in which the violation was detected. This is far from trivial, as e.g. even no input can cause a violation [1].

CMF10: Ability to Quantify the Degree of Compliance. The system should comprise a metric to assess the degree of compliance or non-compliance. The problem with binary degrees of compliance is basically that not every case of non-compliance is critical and should be handled based on its severity. Therefore the approach should have mechanisms to calculate and summarize such degrees.

This can also be relevant for different counter-measurements, as for example, a process can not be further executed by an actor, if the degree of non-compliance is over 0.6 (on a scale from 0 to 1).

2.4 SPRINT

SPRINT (Security in PRocess-Aware INformation SysTems) is a policy definition concept which aims to provide a holistic approach to define contextual security- as well as business rules for PAIS. It links role-based concepts to contextual process and access rules, which are mapped to process models.

The policy design follows the "task based" definition model as defined in "Expression of Security Policies".

It is defined independently to the PAIS or process definition language and describes its context in a mostly abstract manner. This fosters the extensibility of the defined compliance concepts.

Compliance aspects in SPRINT are defined with two different concepts of rules, structural and operational ones. Where structural rules mostly cover task and data based access control aspects, operational rules define contextual process policies. These two concepts are then bound to a role-based access control model.

SPRINT has been chosen as the conceptual model because it is able to include all desired modelling concepts. It enables the definition of business rules and integrates security concepts to its language. Additionally a comparison and an analysis of other existing compliance languages and their expressiveness taken in [2] and shown in Fig. 6 illustrates the potential expressiveness of the SPRINT approach. Hereby SPRINT does not describe the concrete language but rather defines a framework of aspects and connects these concepts to a so called "security policy".

The SPRINT approach mainly consists of three main aspects: structural aspects, operational aspects and organizational aspects. Structural and operational aspects joined together, form a security bundle. This security bundle in combination with organizational aspects (Roles, Units, etc.) then forms one final security policy which is applied to the process.

Structural Aspects. Structural aspects define a basic set of data objects and tasks which occur in the respective process model. Furthermore they define and restrict possible patterns which may occur. Structural aspects are defined independently of the processes control and data flow. During the enforcement of the policies they serve as a basis for the mapping of rules to actual process elements (activities and data) [2].

The structural aspect in SPRINT therefore includes a set of names which refer to the actual process concepts. These elements are called responsibilities,

	AristaFlow / SeaFlows	Apache ODE	Declare	YAWL	SPRINT
Policy Modeling					
Notation	WSM-Net / CRG, FOL	WS-BPEL	LTL	Extended Workflow Nets	PPMEX
Implementation	Task / Repository	Task	Task	Task	Repository
Basic Role Assignment	☒ / ⊡	⊡	☒	☒	☒
Advanced Role Assignment	☒ / ⊡	⊡	□	☒	☒
Policy Constraint Properties					
Basic Data	☒ / ☒	☒	☒	☒	☒
Basic Time	☒ / ⊡	☒	☒	☒	☒
Basic Location	⊡ / ☒	⊡	⊡	⊡	☒
Policy Enforcement and Monitoring (Examples)					
Data	☒ / ☒	☒	☒	☒	☒
Time	☒ / ⊡	☒	⊡	☒	☒
Location	⊡ / ☒	⊡	⊡	⊡	☒
Separation/Binding of Duty	☒ / ⊡	⊡	☒	☒	☒
Policy Verification					
Conflicting Policies	☒ / ⊡	□	☒	☒	⊡
Empty Valid Actor Set	⊡ / ⊡	□	⊡	⊡	☒

Legend:

☒	... supported
⊡	... support with extensions possible
□	... not supported

Fig. 6: Expressiveness of SPRINT, compared to other languages as in [2]

classifying either a data object or a task [2]. Therefore these responsibilities define the corresponding task and data elements which are within the scope of the security policy.

Additionally to the relevant tasks or data elements, an extra data element is defined which acts as a "catch all" element, including all undefined elements [2].

Based on this elements, SPRINT defines: Responsibility Task Pattern Constraints for task responsibilities and responsibility relation constraints for data responsibilities.

Responsibility Task Pattern Constraints. The task pattern constraints describe structural patterns in which the defined responsibilities may occur. In the SPRINT-Paper [2], Linear Temporal Logic (LTL) [16] is used for this description. But it is also mentioned that any arbitrary, eligible language can be used to describe the structural dependencies.

Responsibility Relation Constraints. Apart from the definition of the relevant control flow, the second aspect defines responsibility relation constraints.

These constraints permit or restrict the alteration of data elements used in a process [2].

Responsibility Bundle. The combination of responsibility relation constraints and responsibility task pattern constraints forms a responsibility bundle. These constraints linked together are intended to provide a basis for process consistency checks, as they should reference the whole relevant process scope. Furthermore this combination fosters secure resource allocation of tasks and data to users [2].

Adaptability. As discussed in the Sect. PAIS, structural compliance aspects need to be described in a manner that a change of the process or even during process execution does not harm the validity of the outcome of the evaluation. As in the case of SPRINT, the rule concept is defined independently to the process model and the mapping is only carried out during run-time. Therefore, as long as the process change does not violate any defined rules, a process change does not require an adaption of the defined security policies.

Operational Aspects. Operational aspects compared to the structural rules do not describe the processes semantic or its structure but introduce permissions and constraints for these permissions [2].

Hereby a operational aspect consists of two parts:

- One **Permission**, that defines which operations are allowed in which process model life-cycle stage on a responsibility (task or data object), describing the situation in which the permission is applicable.
- Multiple **Permission Constraints**, constraining the attributes of the corresponding operation.

For the constraint level of a permission, SPRINT describes four different aspects of possible constraints that can be defined [2]:

- Data constraints
  Constraints referring to data, do restrict defined data responsibilities. These can refer to the value, existence or non existence data entries of the defined data element.
- Time constraints
  Time based constraints restrict the occurrence of a task responsibility to a defined time it may occur.

– Location constraints
  Location based constraints do restrict the location for which a operation on a task responsibility is permitted or allowed for a specific resource.
– Separation and binding constraints
  A separation or binding constraint does restrict the assignment of resources (user) to task responsibilities. It either binds or separates the allocation of a task to a certain resource which already obtained the an other corresponding task responsibility. This can only occur in combination with a task pattern constraint.

The combination of a permission with permission constraints composes a permission bundle. This bundle is then applied to a responsibility bundle, as it refers to the responsibilities it defines. Hereby a responsibility bundle can have an arbitrary number of permission bundles referring to it. Yet, a permission bundle or particularly the permission itself applies solely to the defined responsibility bundle [2].

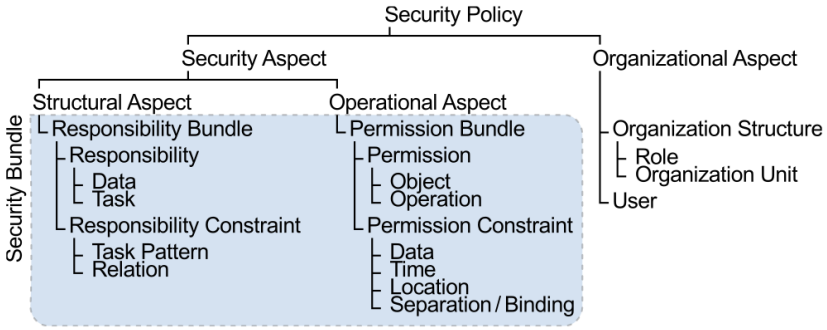


Fig. 7: Overview and connection of SPRINT concepts as in [2]

Combining the Aspects. To assemble the big picture, structural aspects are combined into a responsibility bundle. The elements of the bundle are referenced in the operational aspects, in which all corresponding operational aspects are combined into a permission bundle.

To define the scope and context to which the defined permission bundle refers, it is combined to a responsibility bundle for which is applicable. Therefore it is

defined to which responsibility exactly a permission or permission constraint is referring to.

As shown in Fig. 7 the combination of operational and structural aspects define an overall security aspect. This security aspect, which defines the scope and structure, as well as permissions and constraints for a process. Finally security aspects combined with organizational aspects are brought together in a security policy. For one security policy, a certain security aspect is mapped to an organizational concept (role, units, projects, ...) e.g. a role within an organisation.

There can be more then one security aspect, in most cases there will be multiple. Therefore one organizational concept can have zero to many security aspects applying to it. Also one security aspect can apply to, from at least one up to many organizational concepts.

Scope Compared to CMFF. To define the completeness of SPRINT in contrast to the requirements of the CMFF, an analysis of the functionalities is carried out. This can be seen as an addition to the evaluation which was published within the SPRINT paper, as seen in Fig. 6. This evaluation extends the evaluation carried out in SPRINT in a sense that the evaluation within SPRINT did not distinguish between sub-functionalities of certain concepts. For example the existence of constraints referring to time is evaluated but it is not distinguished if it also includes qualitative and/or quantitative time.

In a broader sense, based on the scope of the compliance framework or concept, it depends on the implementation if and in which form the functionalities of CMFF are present. In most cases this also applies to SPRINT, as it generally only defines abstract concepts rather than a concrete language or implementation guidelines.

Modelling Based Constraints.

- CMF1 Constraints referring to time
  Constraints referring to time are included in the permission constraints of the SPRINT concept. Within SPRINT is it not distinguished which concepts according to time can be defined. These concept could possibly refer to quantitative time (a time frame or concrete time) or qualitative time.
- CMF2 Constraints referring to data
  Constraints referring to data are provided within the permission constraints of SPRINT. There is no concrete distinction between unary or extended conditions, as defined in CMFF [1]. Solely based on the description of SPRINT

this functionality could possibly be omitted, as it is not explicitly specified. On the other hand there are no conceptional limitations which would prohibit the definition of extended data conditions. Additionally within the example constraints of the SPRINT paper [2] a rule with an extended condition is shown. The referenced rule subtracts an data element (date) A from an other data element (date) B and checks if the result is lesser than 3, which in this case is an extended data condition as described in CMFF. When it comes to the distinction between case data and activity data, the SPRINT approach clearly focusses on case data, as data elements (referred to as data responsibilities in SPRINT) are seen globally within the whole instance, and separate to the activities.

– CMF3 Constraints referring to resources
  For constraints referring to resources, SPRINT clearly includes unary and extended resource conditions. Unary conditions insofar that security aspects, as defined in SPRINT, are mapped to organisational concepts which represent resources. Extended resource constraints are included and defined as separation/binding constraints on the SPRINT permission constraint level. It can also be said that the focus of SPRINT strongly relies on this topic of constraints.

Execution Based Constraints.

– CMF4 Supporting non-atomic activities
  The SPRINT approach does not include the definition of non-atomic activities. Nevertheless this aspect can easily be included into the permission constraints.
– CMF5 Supporting activity life-cycles
  The concept of activity life-cycles is not discussed within SPRINT. As the conformance-checking of it can be implemented completely separate to the compliance rule concept, the functionality can be added to virtually any concept as long as the event-stream of the information system supports it.
– CMF6 Supporting multiple instances constraints
  The support of multiple instance constraints is not explicitly considered within SPRINT. But as tasks occur in multiple instances, and may occur even in multiple processes, the structural aspect of SPRINT implicitly outlines a generic of realizing constraints for multiple instances, without additional formal requirements.

User Based Requirements. The scope of user based requirements, as CMFF defines them, is not discussed within SPRINT or rather is not within the scope of SPRINT. This is the case as the SPRINT security and compliance concept primarily discusses the definition of a comprehensive rule and access control concept for PAIS. Aspects of how to handle or mitigate violations are not covered within SPRINT. But with SPRINT as a basis for the rule concepts, most aspects can be included within the concrete implementation.

- CMF7 Ability to reactively detect and manage compliance violations
  This point is split into four sub-aspects:
  - **Detection**, is a fundamental functionality and for that reason violations within the context of SPRINT, can and have to be detected.
  - **Feedback**, based on the SPRINT concept a basic feedback of what (which rule) caused the violation, can be provided.
  - **Continuous monitoring**, as SPRINT is not solely based on model checking or automata concepts, this functionality should be given within an implementation of SPRINT.
  - **Recovery and compensation mechanisms**, to offer these functionalities, an extension of the concept would need to be carried out.
- CMF8 Ability to pro-actively detect and manage violations
  To add the ability to pro-actively detect and manage violations additional concepts would need to be evaluated.

- CMF9 Ability to explain the root cause of a violation
  To explain the root cause of the violation, informations of the structural aspects can be processed in context of the given event stream to determine the existence or absence of a specific event which caused the violation.
- CMF10 Ability to quantify the degree of compliance
  To satisfy this requirement, metric values can be applied to every single operational aspect and as an result summarized to define the degree of compliance for a single instance.

2.5 Process Instance Life-Cycle

The execution of a process instance in every PAIS follows a distinct life-cycle, starting from the creation up to the completion of an instance. Every state within the life-cycle represents an enactment step of the instance.

The detailed knowledge of the underlying life-cycle concepts within the execution engine is crucial for the analysis of the event stream. This is the case as every transition between the life-cycle states represent an action which is carried out during process execution, each including different kind of information.

When talking about a life-cycles, two viewpoints can be considered:

- The **instance life-cycle** represents the state of the overall process instance, the instance life-cycle is passed once during execution.
- The **activity life-cycle** occurs for every activity within an instance.

Several life-cycle meta-models and definitions exist, which primarily focus on the activity life-cycle. To give an overview of different viewpoints on process activity life-cycles, three common life-cycle models are compared.

MXML Life-Cycle Model. The MXML language as defined in "A Meta Model for Process Mining Data" [32], primarily describes an XML language for process logs. As the instance life-cycle is a fundamental part of a process logging language, the publication also includes a basic life-cycle concept. The transaction model as described in [32] is converted into a state model which is represented in Fig. 8.

As displayed in Fig 8, the MXML life-cycle model does not describe life-cycle states but only consist of transactions between different states. Therefore the states "Aborted" and "Completed", as described in Fig. 8 are only assumptions of possible state names. These two end-state names were added to clarify their meaning, as defined in [32].

BPAF Life-Cycle Model. The BPAF (Business Process Analytics Format) concept [10], compared to MXML [32], has its focus particularly on the process life-cycle. Hereby the BPAF model overall defines more as well as more detailed states, compared to the MXML model. Nevertheless it defines itself as an extension to MXML, and its concepts to some extent can be mapped onto the MXML life-cycle model [10].

As displayed in Fig. 9, contrary to MXML in Fig. 8, the BPAF life-cycle model only defines states but does not describe the transitions of these states [10]. The BPAF life-cycle model includes several more states including sub-states. For example a concrete distinction between "Running" and "NotRunning" is made within BPAF, to determine if the instance activity is actively enacted or pending. One important contribution is the further distinction of end-states, to have a better declaration of how the instance execution finished.

XES Life-Cycle Model. Similar to MXML, XES first of all describes a process logging language. XES (eXtensible Event Stream) can been seen as the successor of MXML and aims to solve encountered problems [13]. The IEEE Standard [13] which defines XES, does also references the BPAF life-cycle model as a possible model to display the process activity life-cycle. In contrast to the BPAF model, the XES standard also introduces an own activity life-cycle transitions model, referenced as "standard life-cycle transition model".

The XES standard life-cycle state model as displayed in Fig. 10, shows multiple similarities to the BPAF state model in Fig. 9. The main difference to the BPAF life-cycle model is that transitions between the states have also been defined explicitly. This, according to [13], was the key aspect why a separate life-cycle model has been introduced. The naming schema of the transitions hereby strictly follows the schema of MXML, seen in Fig. 8. Therefore the XES standard life-cycle state model can be seen somewhere in between of the MXML and the BPAF life-cycle model.

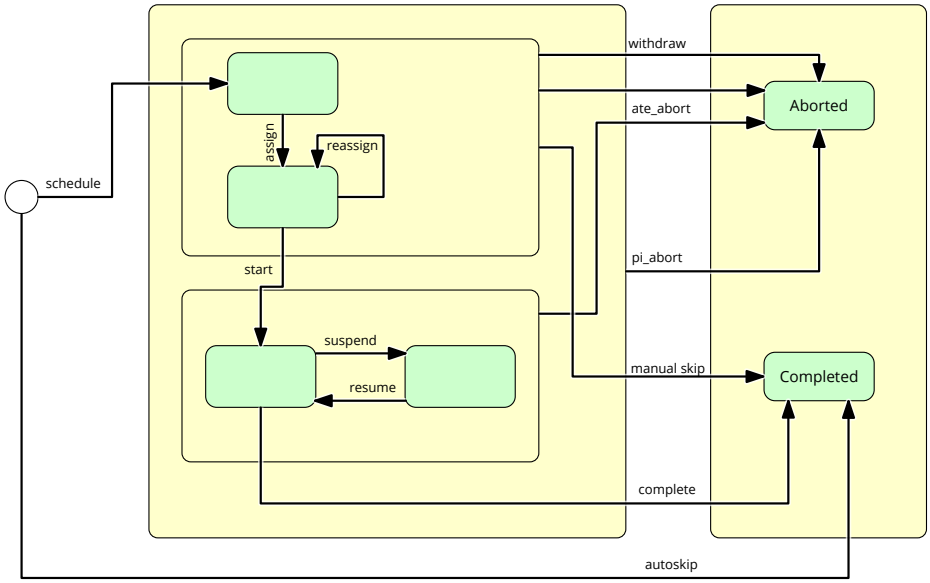


Fig. 8: MXML state model cf. [32]

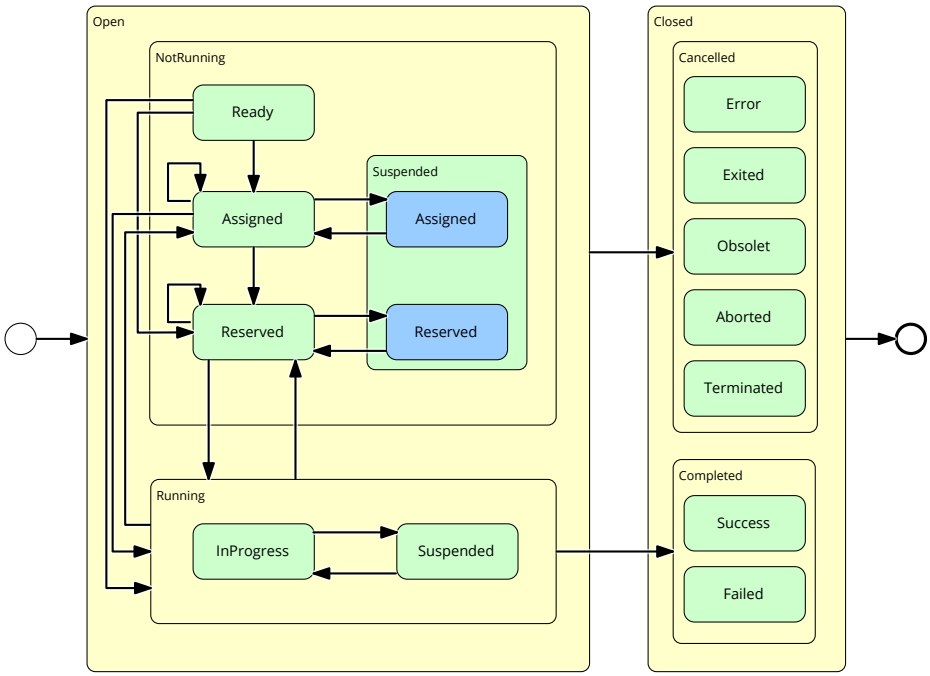


Fig. 9: BPAF state model as in [10]

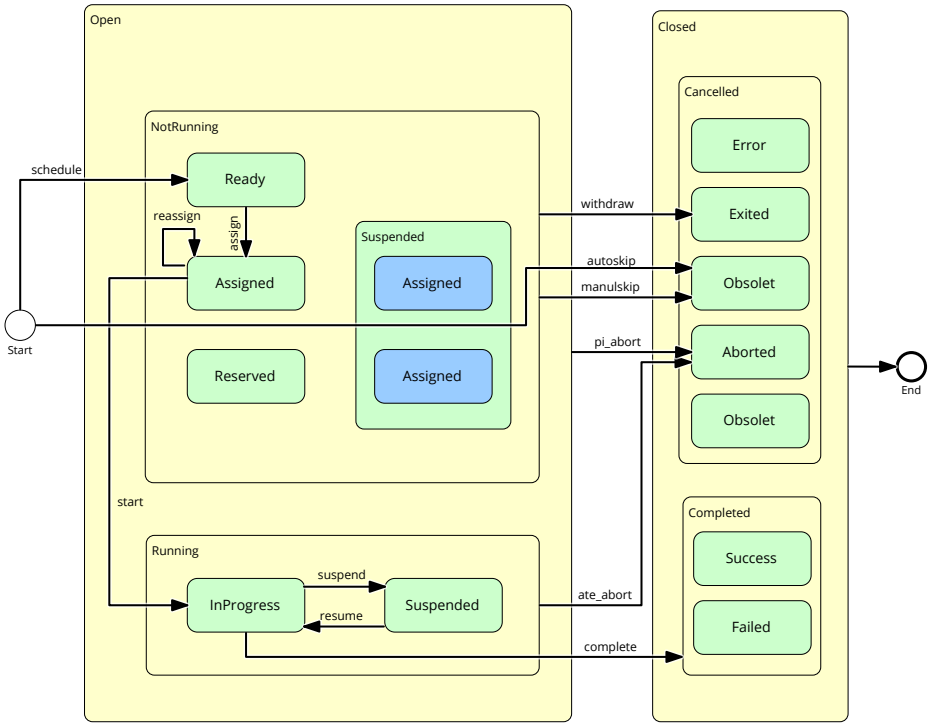


Fig. 10: XES state model as in [13]

3 Requirements and Architecture

3.1 Compliance Aspects Derived

Derived from related work, functional requirements for the Compliance Management system are defined.

Being able to easily extend and change the resulting implementation, was one of the major goals. For example, if the implementation used LTL [20] for determining when to apply a rule, it should be easily replaceable with APQL [33]. Another example for this to-be-achieved flexibility of the implementation: different PAIS use slightly different life-cycle models, thus adapting to different life-cycle models is crucial in order to create an implementation that can be tested against different PAIS.

Furthermore, the system should have the ability to be easily extendable to new requirements and aspects. This includes the possibility to extend the breadth as well as the depth of the rule-sets (see Sect. 2.2).

As defined in 2.2, the compliance monitor should be eligible to handle dynamic process change.

Compliance rules (including business rules and security policies) should be defined separately as this helps to increase the manageability and mutability of compliance rules [9]. This argument is used to justify that the compliance language is a separated from the language that describes the process. It is not integrated into a given process language but rather builds its own grammar to refer to the relevant process flow. This also encourages the ability to separate the compliance monitor from the other involved system components. The result is a lose coupling of the components involved, which again increases the reusability and adaptability of the monitor as well as the other components.

Referring to CMF1: Constraints Referring to Time and CMF3: Constraints Referring to Resources, the key aspect towards expressiveness is to be able to define execution rules which control who can execute or alter which resource, at what point and what moment in time.

To evaluate the proper execution, structural rules can be derived which define a proper order for critical process dependencies, therefore qualitative and quantitative rules as in CMF1: Constraints Referring to Time should be definable.

It must be possible to consider the data elements which are artefacts of the executed process and check them against unary and extended data-conditions

as defined in CMF2: Constraints Referring to Data. To cover security aspects, it should also be possible to include access control rules for data-elements.

For tasks executed within a process, a custom life-cycle model is defined and the correct execution of the activity life-cycle is monitored for every single task, as referred to by CMF4: Supporting Non-Atomic Activities. Additionally CMF5: Supporting Activity Life-Cycles defines that compliance rules can reference this life-cycle states for the definition of conditions.

Besides this key aspects to the compliance monitor and its rule language, the possibility to extend the scope of compliance checking with other aspects should be supported. This especially applies to the remaining CMFF aspects, which are generally considered to be supported but would require the choice of a particular extended approach.

3.2 Underlying Process Engine

The CPEE was selected as the process engine for which the monitoring system is implemented.

> *The CPEE is an open source, versatile, cloud based BPM platform, that has been in development for over several years, and has been used in several national and international projects.* cf. [34]

This Workflow System, which represents the core part of a PAIS is solely process driven. In this terms it represents a "generic system" as described in Sect. 2.1 and furthermore has all characteristics of an "adaptive PAIS" as further distinguished later within Sect. 2.1. Furthermore there are no restrictions to the execution of a process model as far as it is based on common workflow-patterns [35].

The core advantage of the is, that it exposes a very fine-grained life-cycle model as an event-stream, which can be consumed by arbitrary external components. It furthermore is highly customizable and has designated interfaces to manipulate the life-cycle through external components (start & stop instances, delay tasks, ...).

The whole state of the execution engine is discoverable through its RESTful service interface. All communication and manipulation of other services is done based on common web service techniques.

All process activities are based on external services endpoints, which are called upon enactment of the activity. Hereby a service endpoint offers a distinct functionality e.g. to send mails, to fetch the number of disposable goods or to start the fabrication of a product.

Event Stream. The cpee also includes a designated life-cycle model which is strictly followed during execution. There are multiple ways to subscribe to the CPEE's event stream:

- A simple REST request, targeting a specific process instance for which the events are subscribed.
- Defining the subscription within a certain process model, which is then created for every instance of the model.
- Adding a permanent subscription in form of an XML file, which is considered for every process instance of the concerning CPEE instance.

After subscribing, all defined events are then sent to the desired service endpoint, which in this case is the compliance monitoring system.

Listing 1: XML definition of subscribable CPEE events

```
<topics xmlns='http://riddl.org/ns/common-patterns/notifications-producer/1.0'>
  <topic id='activity'>
    <event>calling</event>
    <event>receiving</event>
    <event>failed</event>
    <event>manipulating</event>
    <event>status</event>
    <event>done</event>
    <vote>syncing_before</vote>
    <vote>syncing_after</vote>
  </topic>
  <topic id='position'>
    <event>change</event>
  </topic>
  <topic id='description'>
    <event>change</event>
    <event>error</event>
    <vote>modify</vote>
  </topic>
  <topic id='state'>
    <event>change</event>
    <vote>change</vote>
  </topic>
  <topic id='status'>
    <event>change</event>
  </topic>
  <topic id='dataelements'>
    <event>change</event>
  </topic>
  <topic id='endpoints'>
    <event>change</event>
  </topic>
  <topic id='attributes'>
    <event>change</event>
  </topic>
  <topic id='handlerwrapper'>
    <event>error</event>
    <event>change</event>
  </topic>
  <topic id='transformation'>
    <event>change</event>
  </topic>
  <topic id='handler'>
    <event>change</event>
    <event>error</event>
    <event>debug</event>
    <event>info</event>
  </topic>
  <topic id='simulating'>
    <event>step</event>
  </topic>
</topics>
```

As shown in List. 1, the CPEE offers an XML description including available events to which service endpoints can subscribe to. Apart of events it is also possible to subscribe to votes. Compared to events which broadcast their respective information, votes can manipulate the process execution. Votes are carried out with asynchronous service invocation to subscribed endpoints. After a vote is sent, the CPEE waits for every recipient to answer. Depending on the answers the execution is either continued or suspended, based on the majority of votes. There are several use-cases for the voting mechanisms, as for example, "Rule-Based Synchronisation of Process Activites", discussed in [36].

The subscribable events as in List. 1 are grouped by topics, each defining a different aspect of an instances state and its attributes. As not all of these bits of informations are obligatory for the compliance evaluation a quick distinction for the scopes and meanings of the different topics is carried out:

- ***activity***, includes all events that happen on a single activity life-cycle.
- ***position***, concerns the current step of the execution, changing as the execution proceeds to the next process step.
- ***description***, this event notifies about changes in the process description of the process instance.
- ***state*** events happen, when the state of the whole instance changes.
- ***status***, the status element in the CPEE can hold specific values describing the status of the instance. The status event therefore informs about changes of this element.
- ***dataelements***, describe changes in data elements of the instance. Data elements represent information which is shared across the instance and can be accessed and manipulated by all activities of the process instance.
- ***endpoints***, describe changes or modification of service endpoints of the instance.
- ***attributes***, inform about changes of the instance information. Despite data elements, a process instance also holds information about itself which are called attributes.
- ***handlerwrapper***, represent an additional layer within the CPEE, defining type, structure and information included in the events. Events about changes of these can also be subscribed.
- ***transformation***, the CPEE supports the transformation from multiple process languages into its own DSL. If the transformation schema is changed, a corresponding event is sent.

– ***handler***s in the CPEE are all systems that are subscribed to the events of an instance. Therefore this events notifies about any changes or manipulations of or from the specified handlers of an instance.

– ***simulating***, the CPEE also offers the ability to simulate the process execution. In this case if the execution is only simulated a corresponding event is sent to advertise this behaviour.

As the scope of the evaluation targets compliance and security in terms of process execution not all of the above events are relevant. Events carrying information about the process execution engine itself are therefore left out.

The instance life-cycle, representing the process execution is described in Fig. 11. Green elements represent a distinct life-cycle state and blue elements sub-state to these.

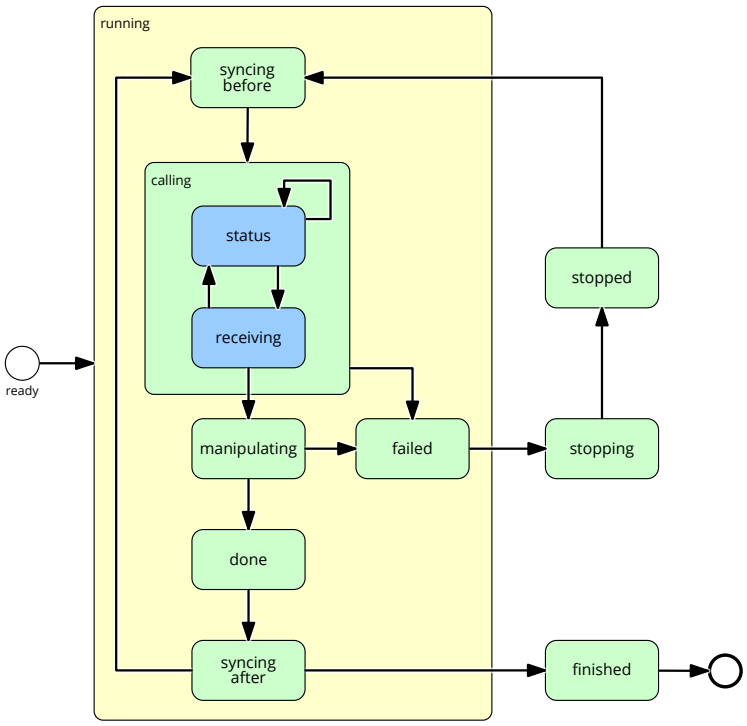


Fig. 11: Activity life-cycle of a single CPEE instance, cf. [36]

Relevant in this case are all events which consider the process instance information. This includes especially the instance activity life-cycle as shown in Fig. 11 and all data elements used during process execution.

Although it is remarked that additional events could be conducted for an extended evaluation of security concerns, this would most likely also include the monitoring of the execution engine itself, which is however not within the scope of this thesis.

3.3 Worklist

Human tasks und thus a worklist are not implicitly included within the CPEE but can be realized with external services. A Worklist in this case represents such an external service endpoint. In case of a Worklist task, the CPEE sends the role-context and the task description to the designated Worklist. The task is thereupon added to the Worklist and can be processed by the worklist users.

In case of the CPEE there is already an implementation of a Worklist with a sufficient role concept which therefore is used for the evaluation.

Listing 2: Subscribable worklist events

```
1 <topics xmlns='http://riddl.org/ns/common-patterns/notifications-
      producer/1.0'>
2   <topic id='user'>
3     <event>take</event>
4     <event>giveback</event>
5     <event>finish</event>
6   </topic>
7   <topic id='task'>
8     <event>invalid</event>
9     <event>delete</event>
10    <event>add</event>
11    <vote>add</vote>
12  </topic>
13 </topics>
```

The Worklist offers a similar interface as the CPEE because it is built upon the same web service technology. The corresponding subscribable topics are shown in List. 2, and are rather self-explanatory. User topics concern all user interaction with the Worklist. All events included into the topic task refer to the creation and deletion of user based tasks.

3.4 Instance Life-Cycle Combined.

The CPEE instance life-cycle, including the Worklist events, as shown in Fig. 12, is able to represent a rather comprehensive view on all relevant activity based events.

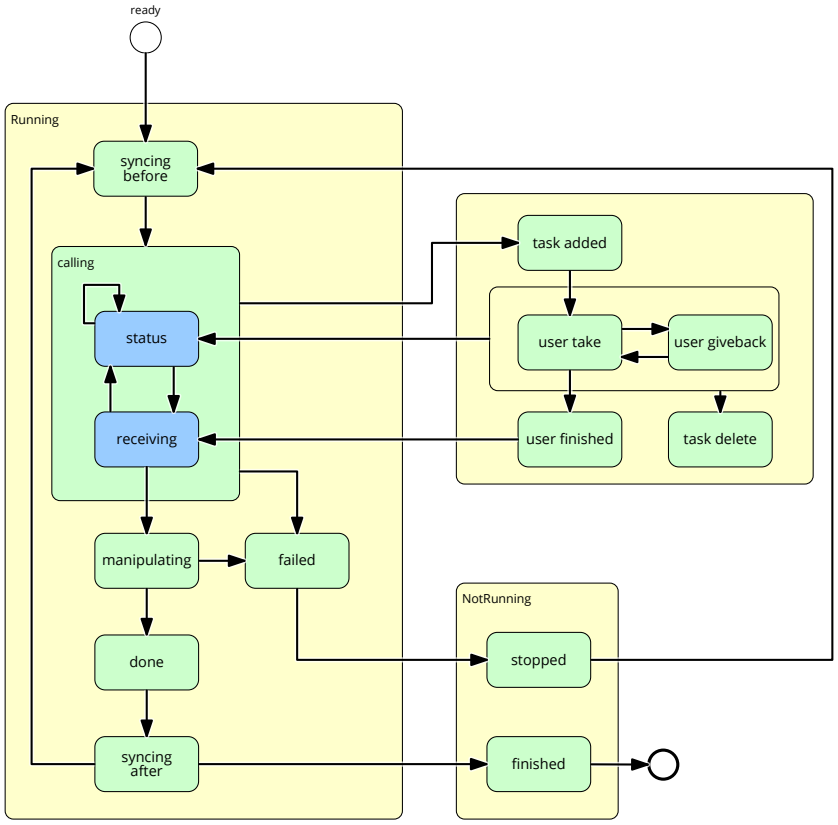


Fig. 12: Instance life-cycle CPEE & Worklist

Compared to the XES reference state model, the state model in Fig. 11 has a greater distinction between different execution states. As depicted in the comparison of Fig. 10 and 11, within the CPEE the “NotRunning” states don’t have as much relevance as in the XES model. Furthermore as apparent in Fig. 12 resource allocation of tasks is outsourced to the Worklist, as it is one of its major concerns.

3.5 Architecture

The architecture consists of multiple modules, shown in the FMC[1] architecture model in Fig. 13. Hereby every module except of COMS already exist, the COMS module therefore is one of the artefacts of this master thesis.

Involved Systems. The core information system structure is based on:

- The CPEE
- The Worklist
- COMS

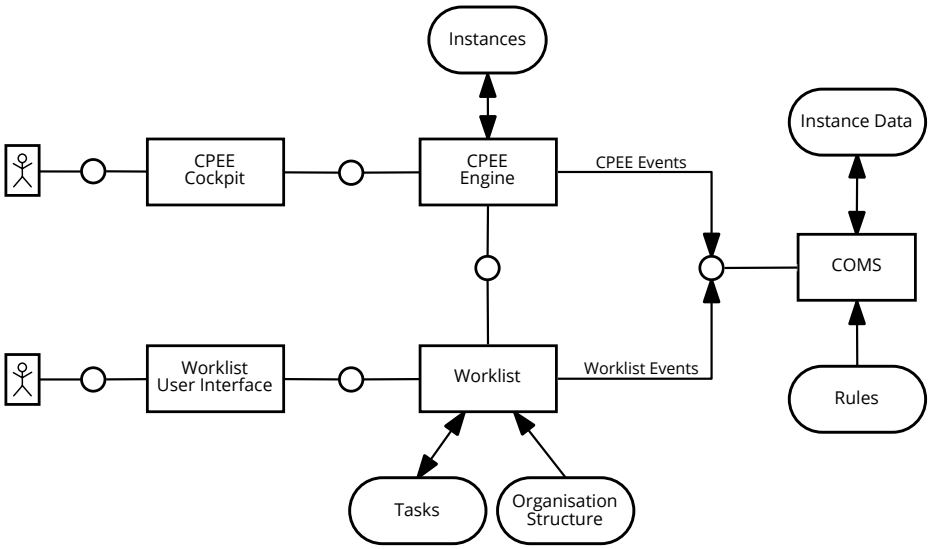


Fig. 13: Communication architecture of involved systems

CPEE & CPEE Cockpit. The CPEE, as described in Sect. 3.2, is the Process Engine. The CPEE itself is accessible through a REST interface and also offers a stateful connection via Websockets. The execution engine enacts process models and manages the process instances.

On top of the CPEE, the CPEE Cockpit offers a web-based user interface. The Cockpit enables users to create, manage and load process models as well

[1] Fundamental Modelling Concepts

as the ability to monitor the execution of CPEE instances. The communication between the CPEE and the CPEE Cockpit is accomplished via Websockets, offering bidirectional communication.

Worklist & Worklist User Interface. The Worklist is based on the same web-service framework and also offers a self-describing RESTful web service as well as WebSocket connections. Similar to the CPEE, the web based user interface operates separated from the Worklist and obtains a WebSocket based connection. The Worklist user interface enable actors to obtain and execute manual tasks.

If a Worklist activity occurs within a CPEE process instance a manual task is added to the Worklist. In a more technical manner, the task is created by a REST based service call from the CPEE to the Worklist. The Worklist therefore holds the data of all tasks and additionally knows the organisational structure on which this tasks are executed. The organisational structure hereby is used to ensure the right resource allocation of tasks. The organisational context is defined within the CPEE instances Worklist activity and communicated within the service invocation to the Worklist.

As shown in Fig. 14, within the CPEE Cockpit, Worklist activities are represented by an avatar symbol and automated tasks are shown as a gear symbol. Even if they graphically represented differently, from the viewpoint of the CPEE both are service invocations.

Fig. 14: Simple CPEE process model

COMS. The COMS engine receives events from CPEE as well as from the Worklist based on the subscribed topics. The possible events and ways to subscribe to this events are hereby discussed in Sect. 3.2. As seen in Fig. 13, the

communication is unidirectional for events. Nevertheless the CPEE REST interface can be used to alter its state and enforce policies. The same applies to the Worklist REST interface. COMS stores the available instance information based on the received events. This information is later used to evaluate the defined rules.

Communication. To exemplify the basic message flow Fig. 15, shows the event stream which is orchestrated by COMS. This sequence diagram considers a whole activity life-cycle of the CPEE with an inclusion of the Worklist. It needs to be remarked that the Worklist does not necessarily need to be included within the execution of an activity. In case of an automated task the internal state is possibly not known because intermediate events as produced by the Worklist would not be given. Therefore information about the resource allocation would be unknown. Nevertheless Fig. 15, shows a comprehensive example of a manual activity.

The message flow in Fig. 15 can be seen analogous to the instance life-cycle diagram in Fig. 12. The sequence diagram hereby describes the message-flow of a successful executed Worklist activity, eventual errors during the execution are therefore left out.

The first as well as the last elements are votes that expect an asynchronous response, which in this case is labelled as vote. Unless it is negative, after the first vote, the activity endpoint is invoked by the CPEE. In this case the endpoint is a Worklist. When the endpoint was called, the corresponding "activity/calling" event is sent to COMS. The "activity/calling" event of the CPEE is followed by a "task/add" event from the Worklist. As soon as a user assigns a task, the "user/take" event happens and a status update is sent to the CPEE. The CPEE then also sends an "activity/status" event to the COMS. This is done on every change of the resource allocation.

Therefore an arbitrary number of "user/giveback" and "user/take" events can happen if the task is reallocated to another resource. This behaviour is expressed by the loop frame. After the allocation of the task finished, the task is processed by an actor and then finally finished. The Worklist then sends an "user/finished" event to COMS and the task outcome to the CPEE.

When the CPEE receives the outcome of the task, the values are processed based on the process description and most presumably data elements of the instance are changed. After the instance manipulation succeeded for this single activity an "activity/done" event is sent, followed by a "syncing/after" vote.

In addition to the message flow, List. 3 shows the structure of a single event which is received by the COMS. The example in List. 3, represents an "activity/calling" event. For an event which is received by the CPEE or the Worklist the basic structure is always the same. The structure always includes:

- A **key** which determines on which subscription the event is based on.
- A **topic**, describing the events topic.
- The **event** type.
- A **notification** which is the payload, including the information of the event in form of a JSON string. The structure and the information included in this payload can strongly diverge, based on the event type. But has a consistent structure within the different events.

Listing 3: CPEE "activity/calling" event

```
{
  "key"          : "coms",
  "topic"        : "activity",
  "event"        : "calling",
  "notification" : {
    "instance":       "http://coms.wst.univie.ac.at:9298/73",
    "instance_uuid": "4abfa6a2-36e4-4c41-943a-b53796f34d08",
    "activity":       "a1",
    "passthrough":    null,
    "time":           1486818321.0265782,
    "endpoint":       "http://coms.wst.univie.ac.at:9300",
    "parameters": {
      "label":  "OK OR NOT OK",
      "method": "post",
      "arguments": {
        "orgmodel": "organisation1",
        "domain":   "Virtual Business 1",
        "role":     "Regular",
        "schaden":  "55546",
        "text":     "fetzen hin"
      }
    }
  }
}
```

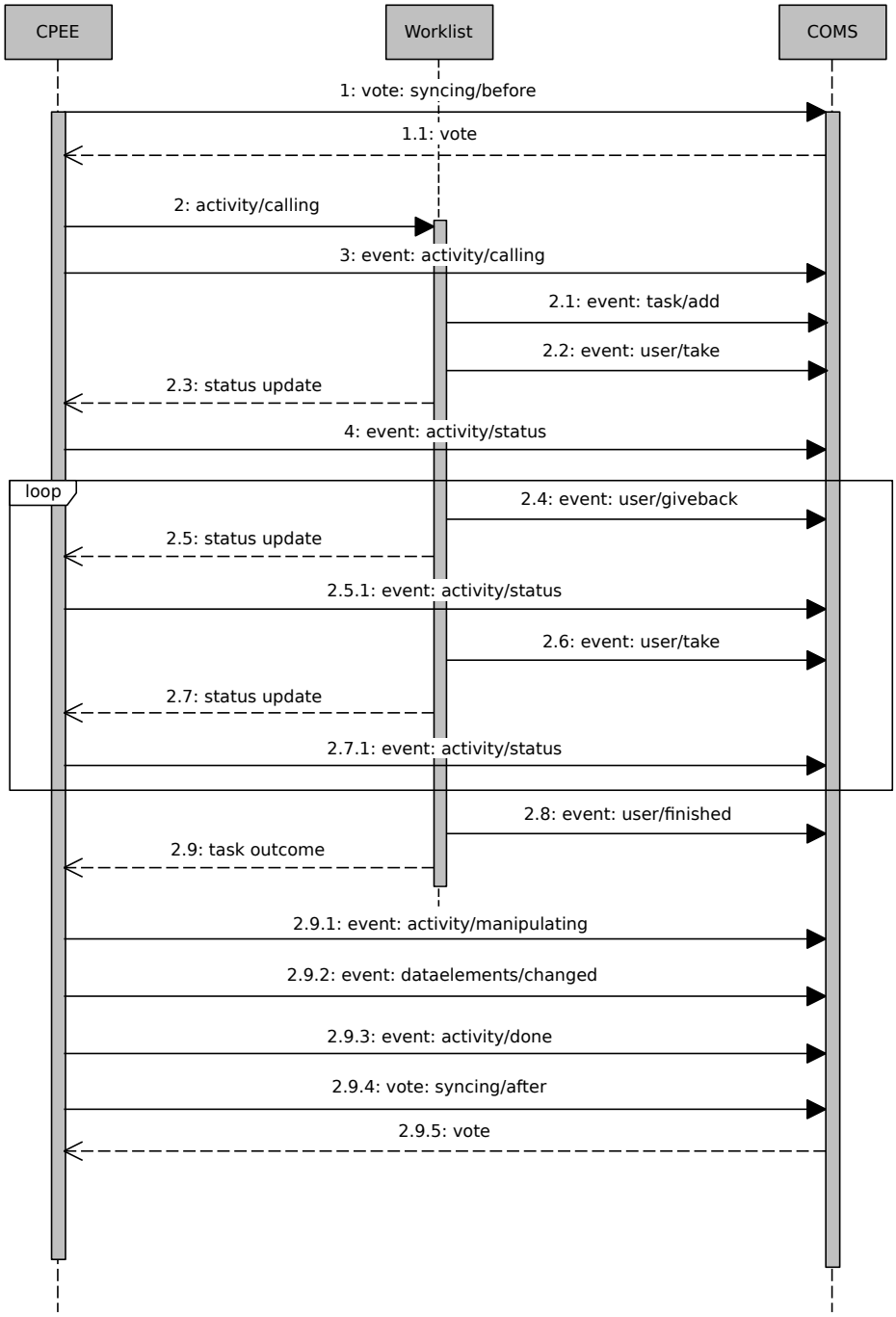


Fig. 15: Message flow CPEE, Worklist and COMS [2]

4 Enforcing Rules - First Implementation

The first approach is merely focussing on the evaluated requirements, and trying to find an architecture which would be able to satisfy those.

One of the basic ideas was to implement a filter concept, in which each filter is checking a different type of constraints. This filters are then applied on events which trigger the specific filter. Every filter type therefore has its own domain, with a set of relevant events.

The Compliance Management system has been implemented by checking the changes, propagated by an event and is discussed in the following section. For compliance checking, all life-cycle states as shown in Fig. 12 have been considered. This also includes the vote concepts "syncing before" and "syncing after", which are also displays in Fig. 12. The vote concepts with its semantic (described in Sect. 3.2) is later used for enactment by altering the process execution, based on the severity of the violation.

4.1 Integration of Constraints

For the evaluation of the compliance constraints the following approach has been selected. Each instance I_x is in a given state $I_{x,s}$, which is kept track of by the implemented rule engine. So an event e leads, for each I_x that it applies to, to a certain state transition $f(e)$ resulting in $I_{x,s+1}$.

$$I_{x,s} + f(e) = I_{x,s+1} \tag{1}$$

Thus when the rule engine receives an event e, two actions have to be taken:

- For each e that is received, each $I_{x,s}$ has to be checked, in order to find out if the e is relevant.
- If e is relevant for I_x, $I_{x,s+1}$ has to be stored.

Therefore the activity entity shown in Fig. 16 only holds the last valid state of the activity, representing the current step of the life-cycle. This decision has been made as for a transition into a new state only the previous state is relevant.

4.2 Mapping Events to Domains

At runtime a major drawback is that not all information is available at any time, therefore the approach in this work is to split compliance aspects in corresponding domains.

Domain is an artificial grouping schema, which allows to group events and rules, so that it is easier to find rules that match certain events. A certain event most certainly does not affect all rules, thus checking to a relevant subset of rules is performance-wise much better, than checking all rules.

As mentioned, not all events are relevant for every domain. For example an "activity calling" event, which is calling an external service, will not include a data manipulation, as this will happen only in a later state of the activity life-cycle.

A mapping, of events causing the evaluation of the respective domains is defined in Tab. 4.

Additional to the in Tab. 4 defined domains, there are specific extended contexts. Instance spanning constraints and organisational constraints.

- Multi instance constraints in this context, reference one of the domains described above but consider more than one instance.
- Organisational constraints add the context of access control concept to elements or events which don't directly include them or for which it is not enforced by the endpoint.

Table 4: Compliance domains and relevant events

Domain	**Description**	**Events**
process scope	This domain does include the verification of all structural evaluation aspects. Included are the execution order or occurrence of activities as well as the execution time, within a process instance point of view.	– syncing before – syncing after
user scope	Including all aspects of resource allocation, as assignment and the execution of tasks.	– activity calling – user take – user giveback – user finished
data scope	Includes the manipulation of data elements used in the instance scope.	– activity receiving – dataelements changed
activity scope	Within the activity scope, state changes in form of events concerning a single activity are checked.	– activity calling – activity status – activity receiving – activity manipulating – activity done – activity failed

4.3 Data Stored

To represent the internal state of an instance, a data model as shown in Fig. 16 has been created.

It is important to note that there is no direct relation between *"Activity"* and *"DataElement"*. For activities the current state and all possible attributes of a single activity within a process are stored. Data is just altered as the result of the execution of an activity or process step (i.e. when the data returned by activities is transformed into *"DataElement"*, some time after the execution). This separation also complies to the specification within 2.4, which also separates data responsibilities from task responsibilities.

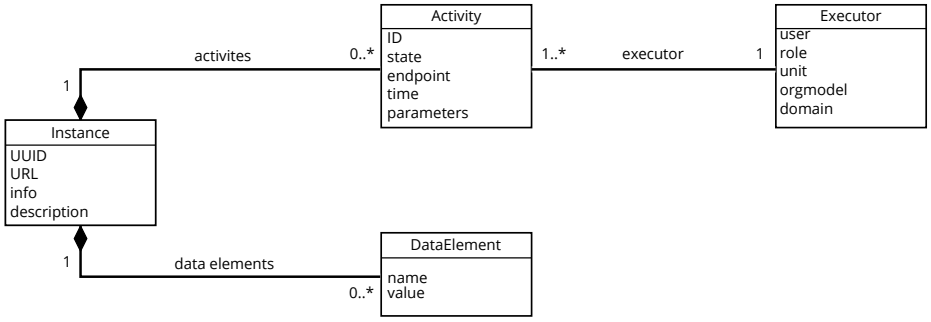


Fig. 16: Data structure first approach

The data structure used to store data for a process instance is represented in Fig. 16. For a given *"Instance"* an arbitrary number of *"Activity"*'s can exist. Every *"Activity"* represents a task in an instance and is identified by its *"UUID"*. The name of the underlying process model is represented in the *"info"* attribute. Apart from the *"id"*, *"Instance"* also holds the process *"description"* (i.e. the whole process model). The process *"description"* can be retrieved as XML in a well defined schema and therefore can be parsed if additional information is needed. The *"URL"* represents the address of the process instance within the CPEE execution engine.

"Activity" and its attributes have been modelled similar to a process activity in the CPEE. In the engine, an activity has a generic identifier ($a1..an$), an endpoint and parameters. The endpoint of an activity represents the service which is called with the defined parameters upon execution of the activity. The *"state"* attribute within the activity entity represents the last recorded life-cycle

state of the activity. The *"time"* attribute determines when the last event, which caused a state change of the activity, has taken place.

The *"Executor"*, representing the organizational model and context of an activity. An activity has exactly one *"Executor"* and an *"Executor"* does apply to at least one *"Activity"*.

4.4 Extraction of Organisational Concepts

Within the CPEE no role concept is present. This is based on the decision to separate Worklist and process engine. Thus for the process engine every task is just a service, which might be a program or a Worklist, which in turn delegates the work to humans.

For "Worklist" tasks, the organisational context is communicated via the parameters sent during the creation of the task. Therefore the CPEE sends a request to the Worklist including a specification of the underlying organisational model for the corresponding task. Upon an activity calling event, the following parameters are used to create a Worklist task:

- **Domain** (required), specifying the domain of the organisational model. Different domains are separated from each other.
- **Orgmodel** (required), represents an URI to the organisational model of the domain, including all entities (units, roles, users) of the domain.
- **Form** (required), the form specifies the artefact which needs to be processed within the defined process task.
- **Unit** (optional), specifies the unit on which the task is applicable.
- **Role** (optional), specifies the role on which the task is applicable.
- **User** (optional), assign the task to a specific user.

In most cases the user parameter is not defined as this would circumvent the whole point of a Worklist.

Therefore a manual task is only a specific type of activity or rather an activity with a specific set of parameters. Not all activities are manual tasks as they can also be automated task in terms of non Worklist service calls.

According to this circumstance, problems with an implementation based on SPRINT arise. As described in 2.4 security aspects are bound to organisational entities. This means that every rule needs a specific organisational entity on which it is applied to. But as described before, this organisational aspects or entities are not present in every activity. The work around in this approach is to introduce an artificial role called "endpoint", which is applied to every task which has no organisational information included. The user in this case is the URL of the service endpoint. With this approach, rules can be applied to service endpoints that only process automated tasks.

4.5 Process Matching

Following the evaluation in Security Policies, the definition of the process activities is strictly separated from the process model. This also means that on evaluation and enforcement of the rules, a mapping has to be carried out. Hereby the defined activity within a certain rule has to be matched to the actual process activity. It has to be remarked that at this step the information of the underlying process model is already present. Meaning that the activities defined in the rule will only be matched to instances created from a certain process model. As stated before, the *"info"* attribute of the *Instance* will be considered as an unique identifier representing a certain process model.

Based on the available informations, different methods to match the process activity with the activity described within the rule can be applied:

- One possibility to match a single activity is based on its generic name (a1..an). The major drawback for this case is that this is incrementally assigned to a activity and does not represent its context. The same generic name can be present in multiple process models, representing a whole different type of activity. This therefore does not facilitate the reusability of defined rules.
- A different approach is to match an activity by its defined service endpoint, as an endpoint represents a concrete functionality of an activity.
- Going even further, also the parameters of an activity are considered. These are relevant as a service endpoint can carry out different tasks based on the defined parameters. The best example for this is the Worklist, based on the parameter "form" different task are carried out by the Worklist user.

Therefore the combination of a service endpoint and the parameters are used to match a single activity. The declaration of parameters is optional, as it is possible that a service endpoint only offer a single functionality and therefore does not need any further specification within the request parameters. If a parameter is given in the rule description, the service endpoint only matches if this parameter is also present in the process model.

4.6 Rules

For the definition of the rules the YAML syntax has been chosen as it is assumed that YAML is more human-readable than other markup languages.

A basic schema definition for the rule-files is given in List. 4. This schema definition is based on the Kwalify[3] syntax.

The section *"Process"* includes the process model name, and all rules of the rule file which apply to the given process. These rules are referenced by the attribute *"id"*, present for every rule under the *"Rules"* section. This rule IDs need to be unique within a rule-file but are allowed to occur in other rule-files.

In the *"Rules"* section multiple rules can be defined. The attribute *"definition"* represents the condition for the rule which is then evaluated. As described in Sect. 4.2, evaluation aspects are split into different domains. To map the rule to the corresponding domain, the *"type"* must be defined. The *"type"* thus refers concept and domain of a rule. Types were introduced to be able to extend the rule-set with different rule conditions. The aspects *"user"*, *"role"* and *"unit"* are relevant for the organizational applicability of the rule. The following cases can be distinguished:

- Unit is defined, the rule is applied to all actors within the defined unit.
- Role is defined, the rule is applied to all actors possessing the role.
- User is defined, the rule will only be applicable if the defined user is executing the task.
- Combination of the above, the rule is only applicable if all defined organisational aspects are true.

The attribute *"operator"* can be optionally defined for comparison operations applicable for conditions within the *"definition"* attribute.

To sum things up, a rule is fetched for validation if:

- The process name matches.
- The defined organisational aspects apply.
- The corresponding type of rule domain is given.

Listing 4: YAML rule schema description using Kwalify

```
1 type: map
2 mapping:
3   "Process":
```

[3] Available at http://www.kuwata-lab.com/kwalify/

```
    type: seq
    sequence:
     - type: map
       mapping:
        "name":
          type:     str
          required: yes
        "rules":
          type:     seq
          required: yes
          sequence:
            - type: int
"Rules":
  type: seq
  sequence:
    - type: map
      mapping:
        "id":
          type:     int
          required: yes
        "user":
          type:     str
          required: no
        "role":
          type:     str
          required: no
        "unit":
          type:     str
          required: no
        "type":
          type:     str
          required: yes
        "operator":
          type:     str
          required: no
        "definition":
          type:     str
          required: yes
```

4.7 Evaluators

The evaluators define the software component which implements the logic to evaluate the condition based on the defined rules. Every evaluator is mapped to a certain domain and therefore only activated if an event of the defined domain is received. For a specific domain, multiple evaluators can be defined.

- **Data Evaluator**: Is evaluated on events within the data scope. It evaluates the data conditions defined in the applicable rule.
- **Structural Pattern Evaluator**: Is evaluated on events within the process scope. Implements the ability to define rules for the process structure.
- **User responsibility evaluator**: Is evaluated on events within the user scope. Checks the assignment of task to resources.
- **Activity life-cycle evaluator**: Is evaluated on events within the activity scope. Correlates all the activity events for a process step and evaluates there soundness.

For the concrete implementation of an evaluator a certain rule-type can be defined, as discussed in Sect. 4.6. For the activity life-cycle evaluator, for example, no rule-type has been defined as this implicitly is checked for all activities of the CPEE.

4.8 Example

To give an example List. 5 shows a possible rule-file defining rules for two process models. The process models defined are *"Akt erheben"* and *"Schadensakt erheben"* representing two similar processes. For *"Akt erheben"* the rules with the ID 1 and 2 apply. For *"Schadensakt erheben"* additionally rule 3 applies.

Listing 5: Rule example

```
Process :
  - name :  'Akt erheben'
    rules : [1, 2]
  - name :  'Schadensakt erheben'
    rules : [1, 2, 3]
Rules :
  - id :              1
    role :            'Regular'
    unit :            '*'
    type :            'responsibility'
    definition :      >
        '(http://coms.wst.univie.ac.at:9300,
        {"form"=>"http://coms.wst.univie.ac.at/form/form-a.html"}),
        (http://coms.wst.univie.ac.at:9300,
        {"form"=>"http://coms.wst.univie.ac.at/form/form-f.html"})'
  - id :              2
    type :            'responsibilitypattern'
    definition :      >
      '(http://coms.wst.univie.ac.at:9300,
       {"form"=>"http://coms.wst.univie.ac.at/form/form-a.html"}),
       (http://coms.wst.univie.ac.at:9300,
       {"form"=>"http://coms.wst.univie.ac.at/form/form-f.html"})'
  - id :              3
    user :            'weissh9'
    role :            'Regular'
    type :            'simpledataconstraint'
    operator :        '<'
    definition :      >
        '(http://coms.wst.univie.ac.at:9300,
        {"schadenssumme"=>"100"})'
```

The process matching does take place within the definition. The in Sect. 4.5 defined matching approach is applied.

- A: (http://coms.wst.univie.ac.at:9300,
  "form"=>"http://coms.wst.univie.ac.at/form/form-a.html")
- B: (http://coms.wst.univie.ac.at:9300,
  "form"=>"http://coms.wst.univie.ac.at/form/form-f.html")

The first value defines the endpoint followed by a key, value pair. The key defines the name of the parameter and the value refers to the actual value the

request parameter must have. If all of this is true for an activity, the rule matches and is applied.

The rules defined in List. 5 have the following semantics:

1. Defines a responsibility for all actors with the role *"Regular"* within any unit. A responsibility rule is a *"User responsibility evaluator"*, and is checked in the user scope. It defines the general rights to execute an activity. The corresponding activities are defined within the definition element.
2. For the second rule no organisational aspects are defined, in this case it definitely is applied and the actor is not relevant. The *"responsibilitypattern"* type is checked within the process scope and evaluates if the process execution complies to a defined order. In this case the definition defines that A implies B. Therefore if B happens, A must have happend at some point before.
3. The third rule defines a data constraint for the user *"weissh9"* within the role *"Regular"*. The constraint restricts that the value *"schadensumme"*, which is the result of a Worklist task, must not be above 100.

4.9 Remarks

For the concrete implementation, the first challenge was to find a proper way to subscribe to the CPEE. In this case the subscription is defined in the process model (different ways are evaluated in Sect. 3.2). This was followed by the analysis of the events. Relevant events for the evaluation were considered and the contained data has been analysed. The analysis showed that certain information was not yet included. One of this extensions concerned the information of data elements. Only the name of the changed data element was included in the corresponding event but no information about the changed values was provided.

The second challenge was that a proper timestamp, representing the time the action took place, was missing.

The third challange was to extract the information concerning the *"Executor"* entity. Based on the activity life-cycle, the information about the organisational context is only included within an activity calling event, as in this step the Worklist task is created with the corresponding organisational context. To be able to check the organisational context of the rule against the context defined in the process, the process description is consulted at the life-cycle state *"syncing before"*. In this case the context can be checked before the activity is executed. Generally the description attribute is used when different process information is not included in events or not yet available. This includes the parsing of the roles and maintenance of the executor information.

The last challenge was to correlate the information obtained from the CPEE with the informations from the Worklist, several adaptions had to be made. This primarily affected the Worklist, to correlate the events to the respective CPEE instance activity the CPEE instance ID, as well as the respective task ID had to be incorporated into every event of the Worklist. After applying this changes, the correlation was a rather simple task.

5 Lessons Learned

The first implementation had one major problem. The information within an event has to be parsed and transformed to fit the internal data format. This makes it prone to errors if the format of a single event changes. Therefore, if certain values of the events get changed, the source code needs to be adapted.

Despite the information that is contained in an event, as described in Sect. 4.9, additional information not included in the event stream is acquired. This causes the inclusion of additional behaviour based on certain events. This means that not all events were processed in a similar way any more. This especially turns out to be true for parsing and extrapolating of data for the "Executor" entity. The "Executor" entity is assembled out of multiple events, every single one including a single part of information, which is extracted and merged into one entity.

The above only represents one certain case where, providing the right data, keeping the internal state consistent was the main problem faced in the implementation. Therefore most of the effort was based on providing sufficient and correct information about the latest state of the instances. Implementing the evaluators themself was rather simple compared to having every information in-place and easily accessible. The validation was straight forward. Apart from the eternal struggle of maintaining a correct internal state, the routing concerning the different domains also tends to be problematic. As the validation was based on the events data, the routing through the program was determined by the events' life-cycle attribute.

For the data structure it turned out to be less than ideal to only store the last state of an activity. For example, the duration between the start of an activity (activity/calling) and the end of it (user/finish or activity/finished) cannot be detected with this approach.

5.1 Findings

For the second implementation, it was clearer which information will be required. Compared to the first approach, the data needs to be saved and represented in a more generic way. This means that changes in the event information do not require an alteration of the internal data structure as well as this should not influence the routing within the application.

Additionally all events should be treated similar, to spare the effort of extensive parsing and transformation of data.

The separation of the domains and the definition of rule types prevents the combination of aspects. Especially the SPRINT defined combination of structural patterns with operational patterns. Therefore at first the structural pattern has to match in the correct order so a specific rule is applicable.

6 Enforcing Rules - Optimized Implementation

"Data first, algorithm second, König last!", Jürgen Mangler 2017

The second approach was based on the lessons learned. It focuses on a more data-centric approach. The information contained in an event is saved the way it arrives and without any manipulation or parsing beforehand. Retrieval of information is hereby realised by querying the collected data. Changes in the event structure don't have to be incorporated into the data structure.

If additional information for compliance checking is required, the information contained in the events is altered. This implies that all the information already is contained within the events. Furthermore every event is stored in the memory and easily accessible. By abstracting the data from the application logic the implementation becomes more generic and is less monolithic.

6.1 Data Representation

The data is stored in a deep, hierarchically ordered, hash structure. The structure is automatically extended, based on the provided information. Therefore no adaptation is required as far as the basic structure holds.

A hash basically consists of a key value pair (e.g. {key1 => value1}), where the key is unique and is pointing to a certain value. This basic structure can be nested by inserting another key value pair (e.g. {key2 => value2}) as value of the first key (e.g. {key1 => {key2 => value2}}). Hereby an arbitrarily deep hierarchy of keys pointing to other keys can be created, whereby the end of the chain always holds a value.

The main challenge in using a hash, is that every key needs to be unique. If for some reason, data is inserted for a key which already exists in the corresponding level of hierarchy, the element is replaced by the newly inserted one. Therefore the hash needs to be designed in a way that no unwanted key collisions occur. To avoid this, a basic structure as shown in Fig. 17 is created.

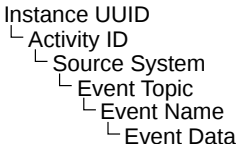


Fig. 17: Basic data structure second approach

The root element of the structure is the instance UUID, which is unique for every process instance. This instance UUID consists of an arbitrary number of activities, represented by their activity ID. Within the next hierarchy level it is distinguished from which source the message has been received from. So far the integrated sources are the Worklist and the CPEE.

The incoming events of this system are inserted based on their topics, followed by the concrete event name. This complies to the structure as shown in List. 1 for the CPEE and in List. 2 for the Worklist. The event name points to the data which is actually retrieved from the event. This event data can have any desired structure and depth.

An example dataset of the data structure, serialized to a YAML format, is shown in List. 6. This example shows data of two independent instances and includes three events. Whereby the *"engine/activity/calling"* events of both instances represent the same activity and both happened within the activity *"a1"*.

Listing 6: Example dataset represented in YAML

```yaml
'08f4aa54-f73d-435a-b15f-a7e354d7706f':
  a1:
    engine:
      activity:
        calling:
          endpoint: http://coms.wst.univie.ac.at:9300
          parameters:
            label: '"OK OR NOT OK"'
            method: post
            arguments:
              orgmodel: organisation1
              domain: Virtual Business 1
              form: http://coms.wst.univie.ac.at/form/form-f.html
              role: Regular
              schaden: '25000'
              text: stark beschaedigt
          time: 1487153642907852467/1000000000
        dataelements:
          change:
            endpoint: http://coms.wst.univie.ac.at:9300
            instance: http://coms.wst.univie.ac.at:9298/80
            instance_uuid: 75997c1c-6008-48eb-bdb8-2b541786f61b
            activity: a1
            changed:
              schadenssumme: 2222
              versnr: 312302021
            time: 7462967401912639l3/500000000
    worklist:
      user:
        take:
          user: weissh9
```

```
32            domain: Virtual Business 1
33            orgmodel: http://cpee.org/~demo/orgviz/
                  organisation_informatik.xml
34            cpee_instance: http://coms.wst.univie.ac.at:9298/78
35            cpee_base: http://coms.wst.univie.ac.at:9298
36            cpee_activity: a1
37            organisation:
38              Workflow Systems and Technology:
39              - Regular
40              - Staff
41            wl_instance: http://coms.wst.univie.ac.at:9300/Virtual
                  Business 1
42          giveback:
43            domain: Virtual Business 1
44            cpee_instance: http://coms.wst.univie.ac.at:9298/78
45            cpee_base: http://coms.wst.univie.ac.at:9298
46            cpee_label: '"OK OR NOT OK" (78)'
47            cpee_activity: a1
48            orgmodel: http://cpee.org/~demo/orgviz/
                  organisation_informatik.xml
49            wl_instance: http://coms.wst.univie.ac.at:9300/Virtual
                  Business 1
50 'b104d78c-fe39-4f2d-a1d7-6af7c14c9ca2':
51   a1:
52     engine:
53       activity:
54         calling:
55           endpoint: http://coms.wst.univie.ac.at:9300
56           parameters:
57             label: '"OK OR NOT OK"'
58             method: post
59             arguments:
60               orgmodel: organisation1
61               domain: Virtual Business 1
62               form: http://coms.wst.univie.ac.at/form/form-f.html
63               role: Regular
64               schaden: '55546'
65               text: fetzen hin
66           time: 7435769119772380l1/500000000
```

6.2 Rule Representation

The rule representation in the second approach is slightly different to the first one. As there is no internal routing of how to process different events, the paths to the information are defined in the rules. Similar to the rule-set in the first approach, the rules are defined in the YAML markup language. The schema definition of the rules is expressed in List. 8. A corresponding example rule is displayed in List. 7.

One rule file, as shown in List. 7 defines exactly one rule for one specific process model. It includes a unique ID and defines the name of the process model for which it is applicable. This design decision has been take for the sake of simplicity, although there is no conceptional reason to remove the restriction on certain processes. Although when all rules are potentially matching all processes, the rule maintenance may become harder.

The *"match"* attribute, as defined in List. 8, describes the structural pattern for which the rule is applicable. This definition of structural conditions for the applicability of a rule was missing in the first implementation and mentioned in the findings in Sect. 5.1. The next field contains the conditions. These are checked if an activity pattern matches. If the whole matching pattern is fulfilled, a statement concerning the compliance of the instance can be made.

Based on the outcome of the evaluation of the conditions, an action is applied. These actions are defined in the *"if"* as well as in the *"ifnot"* part. The *"if"* part is used if the defined conditions has been adhered by the instance. Contrary the *"ifnot"* part is executed if it does not comply to the conditions.

Listing 7: Rule example

```
1 id: 1
2 process: "wltask"
3 match:
4   - :a :
5     - [
6         "engine > activity > calling > endpoint",
7         "==",
8         "http://coms.wst.univie.ac.at:9300"
9       ]
10    - [
11        "engine > activity > calling > parameters  > arguments > form"
              ,
12        "==",
13        "http://coms.wst.univie.ac.at/form/form-a.html"
14      ]
15  - :b :
16    - [
```

```
17          "engine > activity > calling > endpoint",
18          "==",
19          "http://coms.wst.univie.ac.at:9300"
20        ]
21      - [
22          "engine > activity > calling > parameters > arguments > form",
23          "==",
24          "http://coms.wst.univie.ac.at/form/form-f.html"
25        ]
26  condition:
27    - [
28        "a > engine   > activity > receiving > received     >
              schadenssumme",
29        "<=",
30        900
31      ]
32    - [
33        "a > worklist > user     > take      > user",
34        "==",
35        "weissh9"
36      ]
37    - [
38        "b > engine   > activity > calling   > parameters   > arguments
              > role",
39        "==",
40        "Regular"
41      ]
42    - [
43        "b > worklist > user     > take      > organisation > * > +",
44        "include?",
45        "b > engine   > activity > calling   > parameters   > arguments
              > role"
46      ]
47  if:     []
48  ifnot: [cpee_stop, worklist_delete_task]
```

Listing 8: YAML rule schema description using Kwalify

```
1  type: map
2  mapping:
3    "id":
4      type: int
5      required: yes
6    "process":
7      type: str
8      required: yes
9    "match":
10     type: seq
11     sequence:
12       - type: map
13         mapping:
14           str:
15             type: seq
16             sequence:
```

```
17              - type: str
18  "condition":
19     type: seq
20     sequence:
21       - type: seq
22         sequence:
23           - type: any
24  "if":
25    type: seq
26    sequence:
27        - type: str
28  "ifnot":
29    type: seq
30    sequence:
31      - type: str
```

Rule Definition. Within the matching and the condition elements, there exist query like expressions (e.g.: "$a > b > c$), which are used to navigate through the data structure. These are followed by a condition (e.g. "==","$before$","<", etc.), that is accompanied by a clause which may also can be a query.

Query Path. The basic syntax of the queries, described by a regular expression, is as follows: `/^(\w|\s|\>)+\w+\$/`

A sequence of characters is defined by a word as a start, followed by a ">", which operates as separator. This pattern can be repeated multiple times and finally the query must end with a word. Every element between the ">" character is hereby an entity which is looked up. The first entity hereby defines the element with the highest hierarchy, going deeper with each given entity.

As seen in the example rule file in List. 7, the query entities directly refer to the keys in the data structure. In Tab. 5, some queries form List. 7 were selected to illustrate how the queries are applied to the data structure.

Table 5: Rule query structure

source	**event topic**	**event name**	**event data structure**		
engine	activity	calling	endpoint		
engine	activity	calling	parameters	arguments	form
engine	dataelements	change	changed	schadenssumme	
worklist	user	take	user		
worklist	user	giveback	cpee_activity		

As seen in Tab. 5, the data structure of events is referenced at the fourth hierarchy level of the query. This is the case as the first two hierarchy levels are defined to separate the events which occur in different instances and activities. This can be seen as the specialized scope, within the data structure. The other parts are generic and apply to every instance within the same event structure.

Process Matching. The process matching is realised in the same way as it is in the first approach described in Sect. 4.5. Though the general matching concept is kept similar, the definition language deviates from the first approach. Analogous to the general concept, the location of information which needs to match is now represented in a query.

Additionally to the queries, variables are introduced which are valid within the rule scope. If the process matches as described, the activity is mapped to the variable. In case of the rule file example in List. 7, the variables ":a" and ":b" are later used in the conditions section, to define which condition should be applied to which activity.

Currently the following LTL patterns can be expressed:

1. The eventual existence of an activity:

   $$p$$

   This is applicable if only one activity is specified.
2. The existence of an activity:

   $$\Diamond p$$

   The activity must be executed at least once during the process execution
3. The absence of an activity:

   $$\neg\Diamond p$$

   The activity must not occur during process execution.
4. That an activity *"p1"* is followed by an activity *"p2"*. Not immediately but eventually.

   $$p1 \implies p2$$

   Is applicable if multiple activities are defined. Therefore all defined activities are connected by a *"follows"* implication based on the specification order within the rule.

Conditions. After the pattern in attribute *"match"* (see List. 7) fully matches, the conditions for the activity are checked. The condition consist of three parts.

Left hand side: Based on the defined query path, a value from the data structure is retrieved.

Operator: With the operator the condition itself (e.g. "==", "<=", etc.) is specified.

Right hand side: The right hand side is then checked against the left hand side. It can either be a fixed value or also a query path.

Rule queries within the conditions must also include a reference to the matched activities. This reference is represented in the first entity of the rule query. For example the rule query "$a > worklist > user > take$" within List. 7, would reference the activity ": a" defined in match. Therefore the value will be retrieved from the activity which matched the defined pattern in ": a". A rule file can include multiple conditions. Every defined condition is joined with an $\wedge$ operator, which means that every condition must evaluate with "true" for the instance, to comply to the rule.

DSL. For the conditions themself, a DSL is introduced to enable the specification of more complex conditions. Apart from the basic operations: "==", "<=", etc. every other ruby method can be used within the condition field. This in general depends on, if the retrieved value is a string or a numerical object as they provide a different set of methods. More complex conditions can be defined in the DSL and therefore be specified in the rule. Currently the following complex conditions are implemented:

- ***before*** and ***after***
  Based on the time value which is present in every event, it can be checked if the event happens before/after a concrete time event or relative to another event.
- ***withindays*** and ***withinhours***
  Can be defined to evaluate, if an activity has been executed within a certain number of days or hours. This time frame spans from the occurrence of the specified event until it has finished.
- ***tasks_less*** and ***tasks_more***
  These implement an instance spanning constraint, checking if more or less tasks than a given number are currently executed, based on a specific endpoint.

Additionally, to enable the navigation through dynamic information structures, a wildcard operator is introduced in the rule query. An application of this operator is shown in List. 7. By defining the "*" operator, all keys at the next level of the hash will be merged together. If a key of the level, for which the '*' operator is applied, includes a value, the value is added into an array, which is accessible through the "+" symbol. This should only be applied if it can be ensured that the next hierarchy level of the hash, within any of the key, does not include the same key. If a key collision appears, only the last key processed in the merge is retrievable.

Actions. For the enforcement of rules and the amendment of violations, actions can be defined, which are executed based on the outcome of the evaluation. An action is also represented in a yaml file, describing a service call with its parameters.

As shown in List. 22, an action includes a unique name, which is later referenced in the rule-file. The action itself describes request parameters as well as the request method (GET/POST/PUT/PATCH/DELETE). These are followed by the URL, which can also be a query to the data-structure and can be composed of multiple elements. An action can also optionally define concrete request data elements which are sent by the request.

Listing 9: Example for the definition of an action

```
1 - :cpee_stop
2 - :method : 'put'
3 :  url : ["engine > activity > calling > instance", "/properties/values
       /state"]
4 :  data :
5      value : "stopping"
```

7 Evaluation

To conduct the evaluation of the expressiveness and completeness of the approach this thesis relies on the concepts described in Sect. 2.3 as well as Sect. 2.2. Especially all aspects of CMFF, which are relevant to prove the comprehensiveness of the implementation, are highlighted.

Although the implementation sometimes does not cover the full range of a CMFF requirement, the focus of this thesis was to design the architecture of the implementation in such a manner as to always allow for a simple extension to cover the requirement. Such possible trivial extensions are clearly marked as (PTE).

7.1 CMFF Requirements.

CMF1 Constraints Referring to Time. The requirements defined for time constraints include two basic aspects. Qualitative time constraints and quantitative time constraints.

Both types of constraints are supported. For the qualitative aspect the following constructs are possible:

- ***before*** states that, the activity event on the left hand side needs to happen before the activity on the right hand side.
  Example rule:

```
1 [
2   "a > engine > activity > calling > time",
3   "before",
4   "b > engine > activity > calling > time"
5 ]
```

- ***after*** states the same condition as "*before*", but in the opposite direction. Therefore the left hand side needs to happen after the right hand side.
  Example rule:

```
1 [
2   "a > engine > activity > calling > time",
3   "after",
4   "b > engine > activity > calling > time"
5 ]
```

For the definition of the quantitative apect the following constructs are possible:

– ***before*** and **after** also can be applied on concrete time constraints, whereby the concrete time can be expressed by a timestamp or by a complex number. Example rules:

```
1 [
2   "a > worklist > user > take > time",
3   "after",
4   1487153642907852467/1000000000
5 ]
```

```
1 [
2   "a > engine > activity > finished > time",
3   "before",
4   2016-03-08 13:47:55 +0100
5 ]
```

– ***withindays*** restricts the maximal timespan a process activity is allowed to take. The left hand side defines the starting event and the right hand side the maximum number of days.

```
1 [
2   "a > engine > activity > calling > time",
3   "withindays",
4   3
5 ]
```

– ***withinhours*** restricts the maximal timespan a process activity is allowed to take. The left hand side defines the starting event and the right hand side the maximum number of hours.

```
1 [
2   "a > engine > activity > calling > time",
3   "withinhours",
4   6
5 ]
```

CMF2 Constraints Referring to Data. Data constraints, as defined in CMFF, specify two different characteristics which need to be fulfilled to meet the requirements.

Unary and Extended Data Constraints. Unary data constraints compare the information of one domain to a specific value. A corresponding example rule-set in COMS is:

```
1 [
2   "a > engine > receiving > received > schadenssumme",
3   "<=",
4   600
5 ]
```

Here the information of “schadenssumme” within the event domain is retrieved and checked if its equal or less then a specific value (600 in this case).

Extended data constraints compare information of different activities.

```
[
  "a > engine > dataelements > change > changed > id",
  "!=",
  "b >engine > receiving > received > id"
]
```

For the definition of extended data constraints in COMS, both activities must be defined in the “match” section to be referenceable. To define more complex rules multiple conditions can be chained together.

Activity and Case Data. If the data constraints are applied on case or activity data, depend on the specified domain information. Within CPEE the following events are present:

- Activity data:

  ```
  "engine > activity > receiving"
  ```

  This event includes the information which is provided by the activity, representing the outcome of the task.
- Case data:

  ```
  "engine > dataelements > change"
  ```

  This event represents a data change within the process instance specifying which instance data element was changed, and to what value it has been changed.

CMF3 Constraints Referring to Resources. For constraints referring to a resource, the corresponding events must be addressed within the condition. Therefore resource based constraints link to Worklist based events. For resource based constraints, the CMFF also distincts the case of unary and extended conditions.

- **Unary resource constraints** restrict the properties on single resources. Hereby a common example is the restriction of a task if a specific value is exceeded. Hereby an example rule is:

  ```
  [
    [
      "a > engine > receiving > received > schadenssumme",
      ">=",
  ```

```
5       6000
6     ],
7     [
8       "a > worklist > user > take > role",
9       "==",
10      "Manager"
11    ]
12  ]
```

In the example above, tasks with "schadenssumme" greater than 6000 can only be processed by an actor with the role manager.

– **Extended resource constraints** define the related properties of multiple resources.

A common application for extended resource constraints are separation and binding of duties, which are also explicitly addressed within SPRINT.

- **Binding of duties**:

```
1 [
2   "a > worklist > user > take > user",
3   "==",
4   "b > worklist > user > take > user"
5 ]
```

- **Separation of duties**:

```
1 [
2   "a > worklist > user > take > user",
3   "!=",
4   "b > worklist > user > take > user"
5 ]
```

CMF4 Supporting Non-Atomic Activities. The support of non-atomic activities is indeed given within the implementation. Even the activity information is distributed over multiple events.

CMF5 Supporting Activity Life-Cycles. For the support of an activity life-cycle, there is no concrete implementation of a correlation method in the optimized implementation. Despite a concrete implementation, an correlation mechanism can be represented by a single rule-file, using the "*before*" operator. But as it has been done in the first implementation, there is no problem to implement an upstream correlation mechanism which checks if incoming events comply to the corresponding activity life-cycle.

CMF6 Supporting Multiple Instance Constraints. Multiple instance constraints can easily be included into the DSL. This can be done by defining specific

types for multiple instance constraints. To support a broader scope, the language must be extended to determine the scope of the multi instance constraint in addition to the general definition of the constraint.

CMF7 Ability to Reactively Detect and Manage Compliance Violation. The ability to reactively detect and manage compliance violation as defined in CMFF is split into four major aspects.

- Detection:
  Violations in COMS are detected if the pattern is matched but the defined conditions are not satisfied.
- Continuous monitoring:
  If a violation is detected, continuous monitoring is unproblematic within COMS. If a rule is violated, this violation does not interfere with the detection and evaluation of other rules.
- Feedback:
  COMS is able to provide fine grained feedback about which event and what condition caused the violation of the rule. Basic feedback can be advertised via actions, propagating the violation to desired channels.
- Recovery and compensation mechanism:
  Actions can primarily also be used to remediate compliance violations. Hereby multiple manipulation steps to recover from a violation can be defined.

CMF8 Ability to Pro-Actively Detect and Manage Compliance Violation. By now proactive detection was not within the scope of the implementation but a (PTE) noetheless exists. In this specific case the rule has not to change at all, but for the following two cases additional actions (similar to *"if"* and *"ifnot"*) have to be defined:

- **Partial match**
  If one part of a defined match pattern has matched, the following process execution could be restricted. Hereby multiple cases are possible.
  - For the pattern "a is followed by b": if activity "a:" matches, and the defined conditions of a: evaluate as false. The execution of "b:" could be restricted or a specific intermediate event could be sent.
  - If the pattern matches, the values of the respective task are restricted to the parameters in the defined rule.

– **Stochastic rules**
  Based on stochastic analysis of historic process data, preconditions can be defined as rules. Steps as notifying users, altering the process flow or task can talk place to pro-actively prevent a general rule violation.

CMF9 Ability to Explain the Root Cause of a Violation. Within COMS fine grained feedback can be given, describing which condition for which pattern and for which tasks in which rule caused the violation.

CMF10 Ability to Quantify the Degree of Compliance. The degree of compliance is currently only defined by a boolean value. An condition is satisfied or not. But again a simple (PTE) can be found, by introducing an additional attribute in the that quantifies the condition, as exemplified in the following scenarios:

– The arithmetic mean of all complying conditions (1) and all violated conditions (0) can be calculated.
– Severities can be defined for conditions or for activities, defining the overall importance or impact of a violation.

Based on this values a threshold for compliance could be defined. Furthermore different actions based on specific ranges of compliance could be defined within a rule.

7.2 Compliance Rule Patterns.

In the definition of SPRINT and therefore also applied in COMS, rule patterns primarily describe the circumstances under which a rule is applicable. This arises from the main approach of dynamic rule matching which does not consider reference models but describes the relevant pattern in its definition. This leads to a more extensive focus on the rule conditions themself. Therefore in COMS most of the compliance aspects are expressed within conditions and not a temporal logic language. Thus a further evaluation of reasonable compliance patterns can be carried out to define which eventualities can not be expressed within the rule-language by conditions. Based on this evaluation a simple extension of the defined match section can be carried out.

The currently definable match-patterns are listed in Sect. 6.2.

7.3 Additional Requirements.

Apart from the requirements and aspects defined in CMFF during general (see Sect. 2.3), additional objectives have been identified (see Sect. 2.1 and Sect. 2.2):

Adaptive PAIS. Scoped in Sect. 2.1, the compliance monitoring approach must be able to handle dynamic process change. This was achieved by a comprehensive separation of the process and the rule definition. As defined in Sect. 4.5, process activities are solely referenced by their context and no process repository is required. This has the advantage that a rule only needs to be adapted if the characteristics of a referenced activity change. Depending on the match pattern definition this can be the case if:

- The used service endpoint changes.
- The relevant interface description of the endpoint changes.

Altering of the control flow is possible as long as it does not actively violate a given compliance rule.

Depth and Breadth. For the breadth of compliance aspects (see Sect. 2.2) the following (PTE)'s can be pointed out:

- The evaluation of structural elements in form of match patterns and conditions are separated and independently extended.
- The breadth can easily be extend by subscribing to additional events of the CPEE or other systems.

The depth of possible compliance rules, in form of conditions can easily be extended to new or upcoming needs by extending the DSL grammar.

8 Conclusion

8.1 Lessons Learned

Based on the two implementation approaches, a common guideline for a requirements analysis to create an implementation of an Compliance Management system can be derived.

1. **Evaluation of contextual requirements**
   Definition of required functionalities, including desired rule patterns. Based on the required functionalities, an analysis regarding the required informations needed to evaluate the scoped constraints, has to be carried out.
2. **Architecture**
   An analysis of the information system architecture, defining the involved information systems and their context.
   Analysis of the message flow between the involved information systems themself and the monitoring system. This analysis includes the evaluation of:

   - The used communication patterns and technologies.
   - An analysis of how the monitoring system is able to interact with other information systems and how information can be gathered.
3. **Instance life-cycle and information**
   An analysis and definition of the instance activity life-cycle and the involved information. This analysis determines if the required information is present in the message flow of the instance life-cycle and, if not all desired information is present, one of the following decisions need to be made:
   - If possible, include the information into the message flow.
   - If an extension of the information within the message is not possible it needs to be evaluated if the corresponding information can be gathered otherwise.

   If possible no transformation of the data stream should be performed.
4. **Definition of a suitable data structure**
   Definition of a data structure which is able to represent the instance life-cycle and all the required information is key. The algorithms to then implement all necessary functionalities of a CMFF should be rather straightforward.

Based on the evaluated conditions, the implementation of a prototype has been carried out. This was followed by the evaluation of the prototype and,

based on the outcome, further evaluations of the requirements. This follows the methodology described in Fig. 1. One outcome which can be emphasized is that the definition of a suitable data structure is a key ingredient of the solution.

8.2 Future Research

Based on the findings in this thesis the following future research questions arise: (1) How well does the rule-set structure and the data structure comply to different execution engines? (2) Is it possible to deduce a generic event schema for process aware information system including the definition of an instance life-cycle and an corresponding event-stream, capable to include all required information to evaluate common compliance aspects. (3) Is it possible to define what activity and instance life-cycle information is required for the enactment of constraints.

For this data driven implementation, there is less emphasis on temporal aspects. Nevertheless these concepts are essential to express process execution structures. As some patterns are currently included in this approach, others are mostly expressed through conditions. Therefore an evaluation of remaining structural rule-patterns can be carried out. It needs to kept in mind that in this case linear temporal logic is useful for pattern matching. Enforcement of structures might be a use-case which requires a very different language.

References

1. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Information Systems **54** (2015) 209 – 234
2. Leitner, M., Mangler, J., Rinderle-Ma, S.: Sprint- responsibilities: Design and development of security policies in process-aware information systems. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA) **2**(4) (December 2011) 4–26
3. Lewin, K.: Action research and minority problems. Journal of Social Issues **2**(4) (1946) 34–46
4. Ly, L.T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems. Information Systems Frontiers **14**(2) (2009) 195–219
5. Leitner, M., Mangler, J., Rinderle-Ma, S.: Definition and Enactment of Instance-Spanning Process Constraints. In: Web Information Systems Engineering - WISE 2012: 13th International Conference, Paphos, Cyprus, November 28-30, 2012. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg (2012) 652–658
6. van der Aalst et al, W.M.: Process mining manifesto. In: 7th International Workshop on Business Process Intelligence (BPI 2011), Campus des Cézeaux, Clermont-Ferrand, Springer-Verlag (2012) 169–194
7. van der Aalst, W., de Medeiros, A.: Process mining and security: Detecting anomalous process executions and checking process conformance. Electronic Notes in Theoretical Computer Science **121** (2005) 3 – 21 Proceedings of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004)Security Issues with Petri Nets and other Computational Models 2004.
8. Accorsi, R., Wonnemann, C., Dochow, S.: S.: Swat: A security workflow analysis toolkit for reliably secure. In: Process-aware Information Systems. Conference on Availability, Reliability and Security 2011. 692–697
9. Leitner, M.: Security policies in adaptive process-aware information systems: Existing approaches and challenges. In: Availability, Reliability and Security (ARES), 2011 Sixth International Conference on. (Aug 2011) 686–691
10. zur Muehlen, M., Swenson, K.D. In: BPAF: A Standard for the Interchange of Process Analytics Data. Springer Berlin Heidelberg, Berlin, Heidelberg (2011) 170–181
11. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., et al.: Business process execution language for web services (2003)
12. Fette, I., Melnikov, A.: Rfc 6455. Technical report, IETF (2011)

13. van der Aalst, W., Günther, C.: Ieee standard for extensible event stream (xes) for achieving interoperability in event logs and event streams. IEEE Std 1849-2016 (Nov 2016) 1–50
14. Ben-Kiki, O., Evans, C., Ingerson, B.: YAML ain't markup language (YAML) (tm) version 1.2. Technical report, YAML.org (9 2009)
15. Dumas, M., van der Aalst, W., Hofstede, A.: Process-Aware Information Systems: Bridging People and Software Through Process Technology. Wiley (2005)
16. Pesic, M., van der Aalst, W.M.P. In: A Declarative Approach for Flexible Business Processes Management. Springer Berlin Heidelberg, Berlin, Heidelberg (2006) 169–180
17. Reichert, M., Rinderle-Ma, S., Dadam, P.: Transactions on petri nets and other models of concurrency ii. Springer-Verlag, Berlin, Heidelberg (2009) 115–135
18. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: Proceedings of Conference on Organizational Computing Systems. COCS '95, New York, NY, USA, ACM (1995) 10–21
19. Knuplesch, D., Ly, L.T., Rinderle-Ma, S., Pfeifer, H., Dadam, P. In: On Enabling Data-Aware Compliance Checking of Business Process Models. Springer Berlin Heidelberg, Berlin, Heidelberg (2010) 332–346
20. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P. In: Monitoring Business Constraints with Linear Temporal Logic: An Approach Based on Colored Automata. Springer Berlin Heidelberg, Berlin, Heidelberg (2011) 132–147
21. Accorsi, R., Stocker, T.: On the exploitation of process mining for security audits: The conformance checking case. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing. SAC '12, New York, NY, USA, ACM (2012) 1709–1716
22. Ly, L.T.: Seaflows - a compliance checking framework for supporting the process lifecycle (2013)
23. Ferraiolo, D., Kuhn, D., Chandramouli, R.: Role-based Access Control. Artech House computer security series. Artech House (2003)
24. Kalam, A.A.E., Baida, R.E., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miege, A., Saurel, C., Trouessin, G.: Organization based access control. In: Policies for Distributed Systems and Networks, 2003. Proceedings. POLICY 2003. IEEE 4th International Workshop on. (June 2003) 120–131
25. El Kharbili, M., Stein, S., Markovic, I., Pulvermüller, E.: Towards a framework for semantic business process compliance management. Proceedings of GRCIS **2008** (2008)
26. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. Computer Science - Research and Development **23**(2) (2009) 99–113

27. Fahland, D., Lübke, D., Mendling, J., Reijers, H., Weber, B., Weidlich, M., Zugal, S. In: Declarative versus Imperative Process Modeling Languages: The Issue of Understandability. Springer Berlin Heidelberg, Berlin, Heidelberg (2009) 353–366
28. Whitman, M., Mattord, H.: Principles of Information Security. Cengage Learning (2011)
29. Cherdantseva, Y., Hilton, J.: A reference model of information assurance amp; security. In: Availability, Reliability and Security (ARES), 2013 Eighth International Conference on. (Sept 2013) 546–555
30. Park, J., Sandhu, R.: The uconabc usage control model. ACM Trans. Inf. Syst. Secur. **7**(1) (February 2004) 128–174
31. Montali, M.: Specification and Verification of Declarative Open Interaction Models: A Logic-Based Approach. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2010)
32. van Dongen, B.F., Van der Aalst, W.M.: A meta model for process mining data. EMOI-INTEROP **160** (2005) 30
33. ter Hofstede, A.H.M., Ouyang, C., La Rosa, M., Song, L., Wang, J., Polyvyanyy, A. In: APQL: A Process-Model Query Language. Springer International Publishing, Cham (2013) 23–38
34. Mangler, J., Stuermer, G., Schikuta, E.: Cloud process execution engine - evaluation of the core concepts. CoRR **abs/1003.3330** (2010)
35. Mangler, J., Rinderle-Ma, S.: Cpee - cloud process exection engine. In: Int'l Conference on Business Process Management. CEUR-WS.org (September 2014)
36. Mangler, J., Rinderle-Ma, S.: Rule-based synchronization of process activities. In: 2011 IEEE 13th Conference on Commerce and Enterprise Computing. (Sept 2011) 121–128

List of Figures

List of Listings