www.h2020first.eu

FIRST – Project Number: 734599

H2020-MSC-RISE-2016 Ref. 6742023

On-the-fly Service-oriented Process Verification and Implementation

Project Coordinator: Lai Xu, Bournemouth University, UK

With contributions from:
John Kasse, Oyepeju Oyekola, Lai Xu, Paul de Vrieze, Bournemouth University (BU), UK

Revision History	10/08/2020 Layout BU 11/08/2020 First Draft of D4.1 by BU 14/09/2020 Completed Draft of D4.1 by BU

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734599
H2020-MSC-RISE-2016 Ref. 6742023

Table of Contents

1. Introduction

The rapid change in customer demands and intensified competition in the market have pushed manufacturers across the globe to embrace the trend of automation and data exchange in production technology. These intelligent technologies are integrated with traditional manufacturing and capable of collecting and analysing data from every stage of the manufacturing and distribution process to improve efficiency, increase automation, improve productivity, and reduce costs. In addition, they enable new business and service models covering the entire lifecycle of a product. The concept of Virtual Factories vF in the context of manufacturing plays an important role in achieving these goals. vF requires the integration and compatibility of related product design processes and manufacturing processes to support automated design. Notably, the notion of vF goes beyond manufacturing processes for single enterprise processes, as it presents businesses that operate in a global context i.e. Joint collaborative business process (CBP) specified, designed and verified to take advantage of the pool of skills, resources and technology.

By the nature of both collaborative business processes and virtual factories, achieving compliance of CBP in vF setting is complex and requires unique characteristics and requirements due to the design principles for decentralized decision making. For instance, in a CBP, different partners combine resources and skills to design and execute a business opportunity and at the same time act independently from each other. The fact that each partner is geographically distributed makes them distinct from partners in the same location. In particular, besides the internal policies regulations regulating each partner process, their process is also governed by contractual obligations as well as external regulations guiding their overall process operations. Achieving compliance in such a network environment will be challenging and especially when process structure is changed by a partner independently, or when policy or regulations change. For instance, the on-going BREXIT in the UK, once finalized can bring about new laws and regulation, which might have a knock-on effect on CBP across Europe. When these changes happen, it might have a direct effect on the entire business collaboration and its existing processes resulting in significant costs as well as causing an organization needing to modify its entire process or part of it to achieve compliance.

In reality, compliance can be fulfilled before a change in CBP, but this might not remain satisfied after any change occurs. Therefore, after a change has been applied to a process or compliance rules, it is imperative to identify changes in the policy, follow a formal method to re-evaluate the compliance rules, identify the components of the process that are affected by the amendments, and recheck the correctness the process model specifically checking if the existing or redesigned collaborative process still complies with all compliance rules. Falling to manage or constantly monitor these changes can lead to penalties and potential legal issues. However, a knowledge gap exists in supporting the compliance checking of CBPs and the impact of changes in regulation to the business process, with policies beyond control flow to external regulations, laws and standards.

1.1. Contributing partners

The report is based on Deliverable 1.3 as well as PhD thesis of Kasse JP (Kasse, 2019), research papers (Kasse, Xu and de Vrieze, 2017; Kasse *et al.*, 2018, 2020; Oyekola and Xu, 2020). The report is mainly contributed by John Kasse, Oyepeju Oyekola, Dr Lai Xu, Dr. Paul de Vrieze from Bournemouth University.

2. Process Verification and Compliance for Collaborative Business Processes

Figure 1 presents the general relationship between collaborative process verification and collaborative process compliance checking. While the process verification is done at the process design stage, the process compliance check is applied for both process design and process running stages. At the process design time, a business process model is designed to comply with different rules, during process running time; the process compliance is monitored and related according to the changed rules.

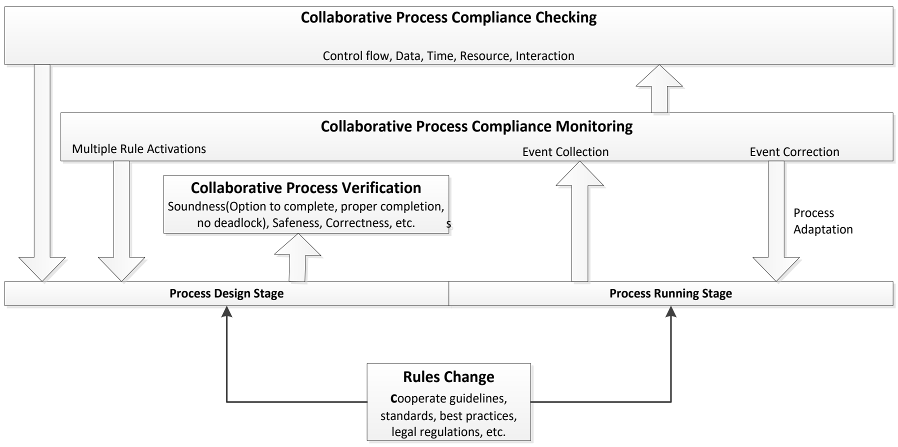

Collaborative Process Compliance Checking
Control flow, Data, Time, Resource, Interaction
Collaborative Process Compliance Monitoring
Multiple Rule Activations
Event Collection
Event Correction
Collaborative Process Verification
Soundness(Option to complete, proper completion, no deadlock), Safeness, Correctness, etc.
Process Adaptation
Process Design Stage
Process Running Stage
Rules Change
Cooperate guidelines, standards, best practices, legal regulations, etc.

Figure 1. The relation between verification and compliance

Naturally, process models may exhibit undesired properties in form of errors which can prevent successful execution. It, therefore, becomes crucial to verify the correctness of the collaborative model before implementation as an erroneous model can result in behavioural anomalies within the entire process. (El Gammal, Sebahi and Turetken, 2014) describes three levels of correctness in process model i.e. (i) syntactical correctness – checking the correct use of element (such as tasks, gateways, swim lanes, events), (ii) behavioural correctness (i.e. business process verification) – are the correctness of a model with regards to a set of properties such as deadlock, livelock, or dataflow errors, and (iii) semantics correctness (Business Process Compliance) – checking whether a business process model complies with applicable standards, law and/or regulations. These levels of correctness interlinked with each other and must be considered to fully achieve error-free model.

To formally verify the correctness of a model, several verification approaches have been identified in the literature as well as different supporting tools. Preliminary works geared towards the verification of syntactical and behavioural correctness i.e. soundness and weak soundness constraints (Robol, Salnitri and Giorgini, 2017). The general approaches of the process verification are based on Petri Net or its variables (transformation) (Governatori and Sadiq, 2009; Knuplesch *et al.*, 2013; Von Rosing *et al.*, 2014; El Gammal *et al.*, 2016) anti-behavioural patterns (Awad, Weidlich and Weske, 2009; Wynn *et al.*, 2009; Roa, Chiotti and Villarreal, 2015, 2016; Knuplesch, Reichert and Kumar, 2017) and direct formalization (Corradini *et al.*, 2017; Houhou *et al.*, 2019). However, despite the several approaches identified in the literature, there are still some gaps in the capabilities of these approaches to verifying the correctness of, in particular, collaborative business processes. For instance, the stated approaches do not fully extend the formalism of complex issues like multiple instances, subprocesses; error handling, loops and data object representation which is important in CBP. Studies that do not abstract data in verification are either required to state-bound domains which are subject to state-explosion or rely on animation than a complete verification (Corradini *et al.*, 2018) making it difficult to verify a model for any possible initial value of the data (Houhou *et al.*, 2019).

While initial works focus on the process verification of syntactical and behavioural correctness, recent work addressed issues relating to semantic correctness (processes compliance). Business process compliance requires checking that business processes are compliant with applicable corporate guidelines, standards, best practices, legal regulations etc. The regulatory requirements are elicited from the general regulatory document and form constraints that restrict the impermissible behaviour of an organization business process. Though, elicitation of relevant requirement from the source document is not sufficient, as the requirements need to be transformed and translated into a formal form i.e. compliance rules to enable compliance verification over business process models.

It is worth noting that business processes compliance relates to conformance to different process perspective that is different phases of the process life cycles such as control flow, data, time and resources perspectives (Borrego and Barba, 2014; Taghiabadi *et al.*, 2014). The different process perspectives help to establish a relationship which facilitates derivation and categorization of compliance rules from the general policies and regulations. Compliance rules must be comprehensible and at the same time should have a precise semantics to enable automated processing and avoid ambiguities. Hence, several researchers have placed their focus on enabling the specification of compliance rules using different approaches with differences in the level of formalism

So far, Business process compliance has received substantial research attention over the last two decades focusing on checking the complaint behaviour of business processes against several policies and regulations. Most of the reviewed studies (Finkel and Schnoebelen, 2001; Goedertier and Vanthienen, 2006; Governatori and Sadiq, 2009; El Gammal *et al.*, 2016; Roa, Chiotti and Villarreal, 2016) mostly focus on the business process within one single organization using different approaches. The approaches focus on different phases of the process life cycle using different languages to support checking in terms of structural behaviour, contractual obligations and privacy and security (Knuplesch *et al.*, 2013). In the same vein, the existing approaches mostly focus on verifying the control flow aspects while abstracting from other process perspectives like data. It is worthy to note that data is a major input for smart devices and machines in a vF supporting automated execution of processes. For instance, with the existing technological innovation comes along with an overwhelming amount of material and product data collected and analysed from every stage of the manufacturing and distribution process and beyond. Particularly, future factories demand a close integration between shop-floor, Manufacturing Execution System (MES) and Enterprise Resource Planning system (ERP) to obtain real-time access to data at production level for real-time execution and vice versa. Hence, it becomes imperative to address other process perspectives, particularly the data perspectives due to its importance in virtual factories settings.

3. Example Use case and Requirements

This section describes an example use case as used in D1.2. It provides in detail the applicable policy and regulatory requirements of the case.

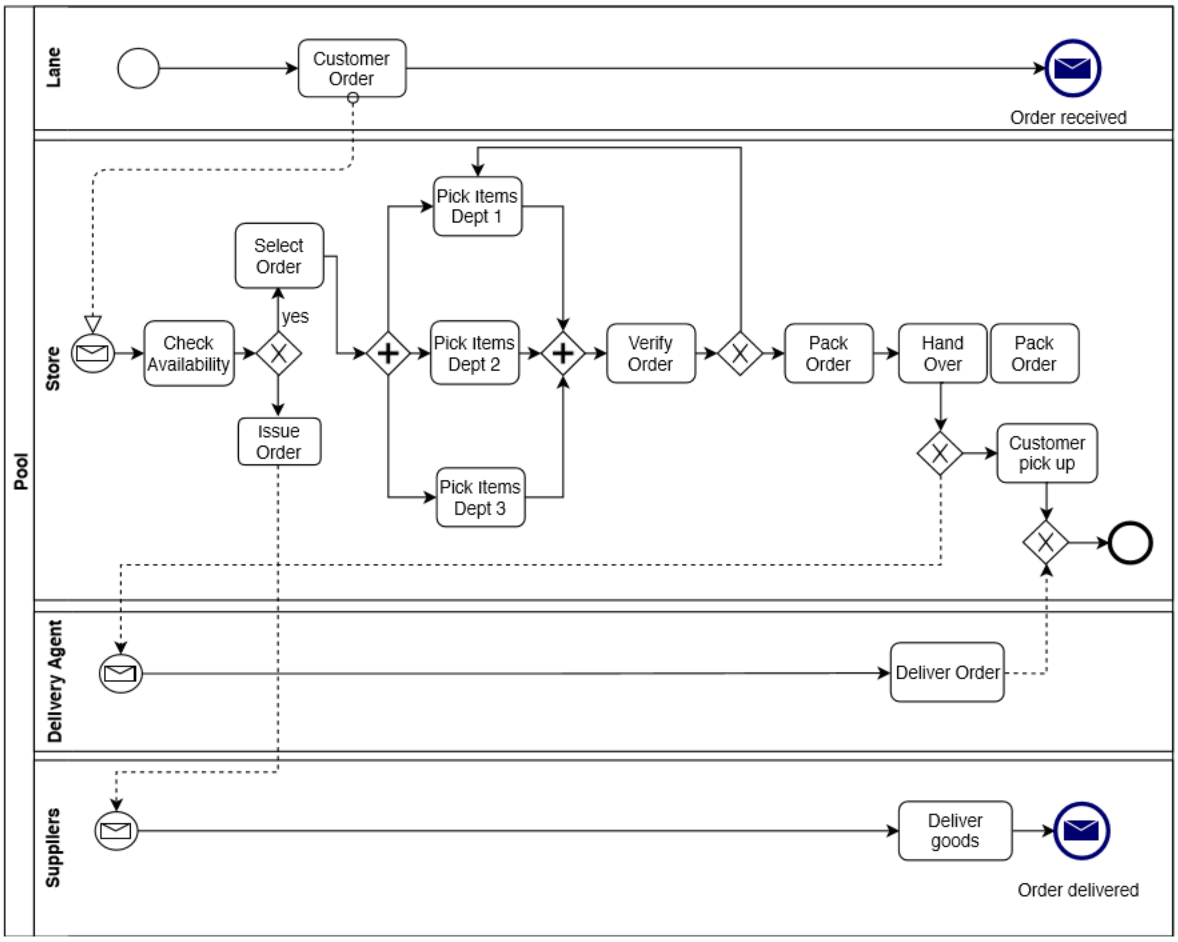


Figure 2: Pick and Pack Use Case

To create orders, customers register on the store's online system. Once a customer order is received, a notification is received at the store while the customer receives a confirmation. At the Store stock levels are checked for item availability. Where stock is below a threshold, a purchase order is issued to the supplier, otherwise, order processing of progress. A staff selects an order, picks items and packs the order. Before packing, the order is verified for conformity with order details, and after it's handed over to customer service. One or more staff may be assigned to an order depending on its size. For items that are out of stock, the order is suspended for a period until stock is available. An item can be substituted with another (for instance, substituting fresh vegetable items with tinned vegetables). Supervisors can contact customers to seek opinion either to wait, change or cancel an order in case items cannot be substituted. A customer can cancel an order delayed beyond a specific time. Ready orders are either picked up by the customers or home delivered by the store. For further understanding, the following assumptions are made: The process model is adopted by stores of different size and capacity; Stores are in different regions where different laws and regulations apply; Stores vary the model to suit local policies.

The above pick and pack business process (displayed in **Figure 2**) is subject to different applicable policies and regulations which form constraints restricting it to specific behaviour and determine how the operations are been conducted. The applicable policies are amalgamated into a set of policy requirements as presented in **Table 1** below. The policy requirements are categorized based on the different process perspectives that are control flow, data flow, resources and time-based.

Categories	Constraints
Control flow	• Some activities can be combined and executed together depending on store size. E.g. Pick items and pack items • Pack Order immediately follows verifies order. Ready orders are either picked by the customer or delivered by the store • Delays are communicated to customers. • Pick items are repeated until all items are picked. • Notify customer order details immediately after submission
Data Flow	• Customers register on the system before creating orders. • Users must be authenticated to access system. • System must be up to date with all relevant data. • Customers can track their orders via the System. • Access to customer data is restricted by privacy constraint. Bulky orders e.g. with orders above £5000 can pay by cheque
Resources	• Resources are assigned to tasks. • Resources must be uniquely identified and authenticated. • Where resources are assigned work based on shifts, access to data is also based on shifts. • Resources like packers and pickers are binding of duty constrained. • Some resources like pickers are restricted from executing some tasks e.g. verify the order. • Some resources like Verifiers are Separation of duty constrained. • Some tasks like Handover order can be delegated.
Time-based	• The system must be available 24/7. • Each task is time-bound and the total process duration is aggregated from task durations. • Some tasks can be delayed for not more than one hour. • Resources are allocated according to time shifts e.g. dayshift or nightshift

Table 1: General Policy Requirements

The internal policies stated above are established as operational guidelines, which may vary from one store to another to suit specific requirement provided that there are no violations to the reference policies or regulations. Besides the internal policies, some relevant external regulations apply to the process that is:

- The Sarbanes Oxley Act and Base III with requirements for separation of duty and biding of duty.
- The GDPR with requirements for security and data privacy.
- The consumer protection Act 2015 UK specifies service level requirements to protect consumers.

- NIST-National Institute of Standards and Technology

Compliance documents are written in natural languages with associated ambiguity. The ambiguity leads to false interpretations, misunderstandings and confusion. Moreover, regulations are stated in a prescriptive manner, i.e. they specify what is required but are silent on how it should be achieved. As a consequence, organisations have perceived compliance as a tedious burden and a complex task, especially where skills or automated compliance tools are not available to support enforcement and verification. This report presents a compliance approach to support: (1) the elicitation of new compliance requirements arising out of the changes in policy and regulations, (2) Formalisation of the requirements into compliance constraints and (3) Checking and verifying compliance of the business process with the constraints.

The first step in achieving compliance involves the ability to identify the relevant policies and regulations that the process model should conform to. These are derived from the relevant regulations as stated earlier. The identified changes are formalised and used to check the business process for conformity. Changes committed in the process can be a source of non-compliance. Checking the impact of policy and regulatory changes over the business process is essential to determine its compliance. Verification techniques like theory proving, model checking, and simulation are used to achieve verification. Hence, this report presents how a simulation technique is used to assess the impact of changes in policy and regulatory requirements over the performance of the business process.

4. Expression and Specification of Compliance Constraints

This section presents concepts related to compliance requirements, their expression and translation into constraints. It presents an illustration of elicitation and categorization of requirements based on the requirements from the use case presented in **section 3**. To express and represent the compliance requirements, a Descriptive Logic is introduced and used, whereas the translation of requirements into constraints is enforced by the integration of Description Logic (DL) and Linear Temporal Logic (LTL) to achieve formalised constraints to which reasoning can be applied to achieve compliance verification.

4.1. Constraints

Constraints restrict processes to specific behaviour as required by policies and regulations. Without the restrictions, models would execute any desired behaviour or end-users would be at liberty to undertake any desired operations. However, the existence of constraints creates restrictions on process behaviour. A constraint is a rule prescribing behaviour as conditions that a business process must conform with. Mandatory constraints must be satisfied by the execution of the model otherwise a violation occurs. 'The optional constraints may be satisfied or not, their lack of satisfaction does amount to a compliance violation. 'The choice to execute activities with optional constraints may depend on the availability of time, computation complexity requirements like time or added value that may be derived from the business or customer. During process execution, the constraint conditions become active and are evaluated by the process engines to guide further execution. Compliance is achieved when the constraint conditions are fulfilled i.e. the outcome of the process behaviour matches the prescribed behaviour.

4.2. Constraints for Collaborative Business Processes

Based on the characteristics of collaborative business processes, which are bound to comply with constraints from various external regulations. In this section, a formal structure of categories of constraints binding to collaborative business processes is summarized in **Figure 3** based on constraints categorizations described in (van der Aalst, ter Hofstede and Weske, 2003; van der Aalst, 2004; Weske, 2007)

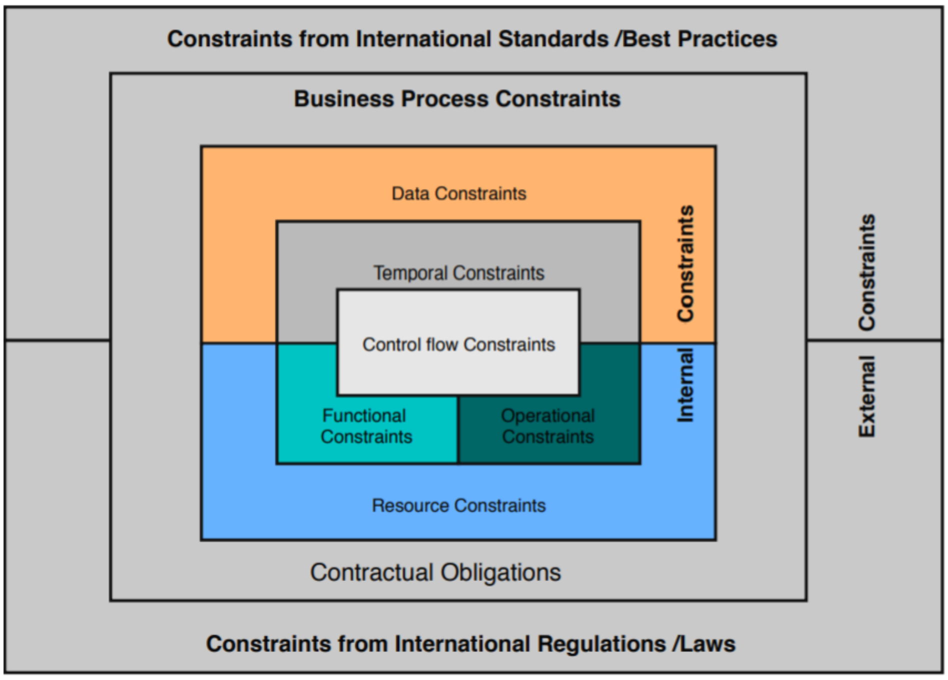


Figure 3: Constraint categories based on Structural Perspectives of the business process

Figure 3 is a constraint relationship model illustrating logical relations between collaborative business process constraint categories. At the core of the model lie the control flow constraints which form a basis for all other internal constraints. It borders with the temporal, functional, operational, resource and data constraints which all form internal constraints. Beyond the internal constraints are contractual obligations; these integrate policies from partners. Next are constraints originating from the external regulatory agencies outside the business environment. External constraints are regulations, standards, best practices and laws that to regulate the behaviour of the process beyond its borders or contractual obligations. Example categories based description of the constraints follow:

Pattern /condition	**Description**	**Purpose for checking**	**Example**
Existence	An activity must occur in an instance or otherwise	Occurrence or absence of an activity	Activity C must exist in every instance of the process. E.g. every order must be verified
Bounded Existence	An activity must occur for a specific number of times	Multiple occurrences of activities	Activity B executes several times until a required condition is fulfilled. E.g. the pick items activity is repeated until all items are picked.
Dependency	Execution of an activity based on occurrence or non-execution of another	Occurrence or absence of dependent activity	For activity C to occur, activity B and A must-have executed successfully or otherwise. E.g. shipment of the goods depends on confirmation of payment
Parallel	A set of activities must occur in parallel	Activities that are bound to occur in parallel	Activities C and D are mutually exclusive. E.g. upon order confirmation, an invoice is sent to the customer while the order is being processed.
Bounded Sequence	Number of times a chain of activities must occur	Number of occurrences of chained activities or otherwise	Activity B and C execute several times until the required condition is fulfilled
Precedence	An activity must occur before another. ‘is also true for chained precedence for limiting a chain of activities	Order of occurrence i.e. activities that must occur before other(s)	Activities A and B are followed by C. E.g. every account balance checking is preceded by successful Login of the account holder
Response	An activity that must occur due to the occurrence of another. ‘is also true for a chained response for limiting a chain of activities	Order of occurrence i.e. activities that must occur after other(s)	Activity E will occur if activity C occurred. E.g. payment by cheque activates the cheque processing activity

Table 2: Control Flow Constraints Categories

Constraints	Description	Example
Data Visibility	Definition of data elements based on the structure of the process and scope of accessibility of data	*Task data*: Describes data elements accessible by the task or by each of the components of the corresponding tasks blocks. *Scope Data*: Data elements defined which are accessible by a subset of the tasks in a case or defined according to several tasks that are coordinated. *Multiple Instance Data*: Tasks that occur multiple times in a case can define data specific to an individual execution instance *Case Data*: Data accessible by all components during the execution of the case. *Workflow Data*: Data elements are accessible by all components in each case of the process and its context. *Environment Data*: Process components have access to external data
Data interaction i.e. internal & external	Definition of data elements based on how data is exchanged between process components and how their characteristics determine data flow	Data Interaction between Task to task Block Task to Sub Workflow decomposition sub-Workflow Decomposition to Block Task Multiple Instance Task
Data validity	Definition of data in a state that is useful and meaningful for task execution	Controls necessary to keep data up to date
Data availability	Definition of data in a format that is ready for use and application	Defining which data has universal access and ensuring its universal availability
Data accessibility.	Definition of data elements that make data accessible.	Access control and authorization, regulation and legitimization
Data privacy	Definition of data elements that preserve the privacy of data	*Permissions*: Represents a set of permissions granted to users to access data such as Permitted, Forbidden, and permitted if the condition is true. Permissions are linked to actions performed only if one has permission to do so. Also, the purpose is linked to permission; permission cannot be given unless a purpose is specified. It is also used to represent user consent. *Conditions*: Conditions that must be true to allow an action to be performed on data. *Data Retention*: Defines the period data is kept at the requester end
Two – Three – Four-way matching	Requires values of two different data objects to match	Received goods must match the payment invoice.

Authenticity	Requires Identification management for controlling the data access	All users are identified and authenticated by the system

Table 3: Data Flow Constraints Categories

Constraint	Description	Example
Instance duration	Instance duration	A set of activities from *a*. . . .*n* must be executed within one (1) hour from the time execution starts. E.g. the processing of an order should last for one (1) hour from the time the order is submitted.
Delay	Period within which an activity can be delayed	Activity *b* will execute after exactly two (2) hours once execution of activity *a* is complete. E.g. after customer payment, shipment will be delayed for two (2) hours until payment is confirmed or reflected on the system
Validity	Period within which an activity can be executed	Activity *b* will execute between 12:00 and 14:00 hours every day of the week. E.g. the shipment of goods takes place between 12:00 and 14:00. Or customer care service is available only during normal working hours.
Duration	The period for which an activity is scheduled to execute from start to completion	Activity *a* will execute for 45 minutes. Or activity *a* execution takes between 20 and 50 minutes
Repetition	The period between which an activity can be repeated	After initially failed execution, activity *a* can be repeated for twice, otherwise, it is restarted after 1 hour. E.g. log on can be tried for three (3) successive times, if it still fails it is restarted after one (1) hour
Overlap	Period within which an activity can start and complete regarding another activity's start and completion period.	Activity *b* is scheduled to start 30 minutes after activity *a* has started but can complete together, however, *b* should not complete before *a* completes. E.g. pack items can start 3 minutes after verify order has started but cannot complete before verify order completes.

Table 4: Temporal Constraints Categories

Constraint	Description	Example
Segregation of duty	Requires separate execution of high-risk tasks by different actors	Cheque processing is executed by two different actors.
Binding of Duty	Requires 2 or more related tasks to be executed by the same resource.	The Doctor who diagnoses a patient must also prescribe drugs.
Delegation	Share or transfer permissions and associated responsibility from one actor to another.	A supervisor can delegate verify order to pickers.

Table 5: Resource Constraints Categories

Tables 2 to 5 present set of general compliance patterns as observed from the literature (Dwyer, Avrunin and Corbett, 1998; Wong and Gibbons, 2011; El Gammal *et al.*, 2016; Ramezani Taghiabadi, 2016). For example, **Table 2** presents example control flow constraints patterns including existence, bounded existence, dependency, precedence, and response inter alia. **Table 3** describes data constraints with examples. Data constraints include data visibility, data validity, availability and accessibility, privacy among others. **Table 4** presents temporal constraints described with examples namely Duration, instance duration, delay, validity among others. In **Table 5,** resource constraints are listed with examples given including separation of duty, binding of duty and delegation. Lastly, because temporal constraints do not exist independently, **Table 6** is a matrix matching temporal constraints with other constraints to benefit a combination of constraints during their expression and specification. In subsequent sections, the constraint patterns are described formally by deriving logical relations using description Logic and LTL.

Temporal constraints	**Delay**	**Validity**	**Duration**	**Repetition**	**Deadline**
Control flow	Existence Precedence Response	Existence	Parallel	Dependency Bounded Existence Bounded Sequence	Existence Precedence Response
Resource flow	Authentication	Authentication	SoD Binding of Duty	Two-three - Four-way matching	Authentication Privacy SoD BoD
Data flow	Data accessibility Data validity	Data availability Data accessibility Data availability Data visibility Data privacy	Data interaction	Data accessibility	Data accessibility Data availability Data visibility Data privacy.

Table 6: Temporal Constraints and Combinations with other Constraints

4.3. Constraint Expressions

In literature different forms of logic are used to define, express and specify constraints. For instance, studies by (Pesic, 2008; El Gammal *et al.*, 2016; Groefsema, 2016; Ramezani Taghiabadi, 2016) use different forms of logic that compose their proposed languages. However, this formalism remains difficult to comprehend by ordinary end users like compliance officers and other stakeholders. This is especially significant for stakeholders who are the subject matter experts yet lack technical knowledge of defining and specifying constraints. Hence, the application of descriptive logics (DL) is generally adopted and adapted as less complex constraint expression formalism, upon which we base to compose a mechanism to express and specify constraints. 'The motivation to use DL is based on its rich syntactical and semantical vocabulary, which is yet easy to understand and use by ordinary users due to its closeness to natural language. Constraints expressed and specified in DL are easy for human intuition, understanding and interpretation. Besides, DL remains expressive enough to support reasoning over constraints and their eventual checking.

4.3.1. Description Logic

DL is a language used for formal representation of knowledge by facilitating formal expression and specification of requirements of knowledge base systems. DL extends into different types like spatial, temporal and fuzzy logics with different features to support various forms of expressivity and reasoning complexity. Here, we highlight concepts relevant for application to policy and regulatory requirements definition. In **Figure 4**, a set of DL applicable syntax and semantics are given.

Constructor	**DL Syntax**	**Semantics**
Universal, top	$\top$	Δ^I
Bottom	$\bot$	$\emptyset$
Intersection	$C_1 \sqcap C_2$	$C_1^I \cap C_2^I$
Union	$C_1 \sqcup C_2$	$C_1^I \cup C_2^I$
Negation	$\neg C$	Δ^I / C^I
All values from	$\forall P.C$	$\{a \in \Delta^I \mid \forall b.(a,b) \in P^I \rightarrow b \in C^I\}$
Some values	$\exists P.C$	$\{a \in \Delta^I \mid \exists b.(a,b) \in P^I \wedge b \in C^I\}$
Max cardinality	$\leq nP$	$\{a \in \Delta^I \parallel \{b \in \Delta^I \mid (a,b) \in P^I\}\mid \geq n\}$
Min cardinality	$\geq nP$	$\{a \in \Delta^I \parallel \{b \in \Delta^I \mid (a,b) \in P^I\}\mid = n\}$
Qualified at-most restriction	$\leq nP\ C$	$\{a \in \Delta^I \parallel \{b \in \Delta^I \mid (a,b) \in P^I \wedge b \in C^I\}\mid \leq n\}$
Qualified at-least restriction	$\geq nP.c$	$\{a \in \Delta^I \parallel \{b \in \Delta^I \mid (a,b) \in P^I \wedge b \in C^I\}\mid \geq n\}$

Figure 4: DL Syntax and Semantics

4.3.2. Control Flow Constraint Expressions

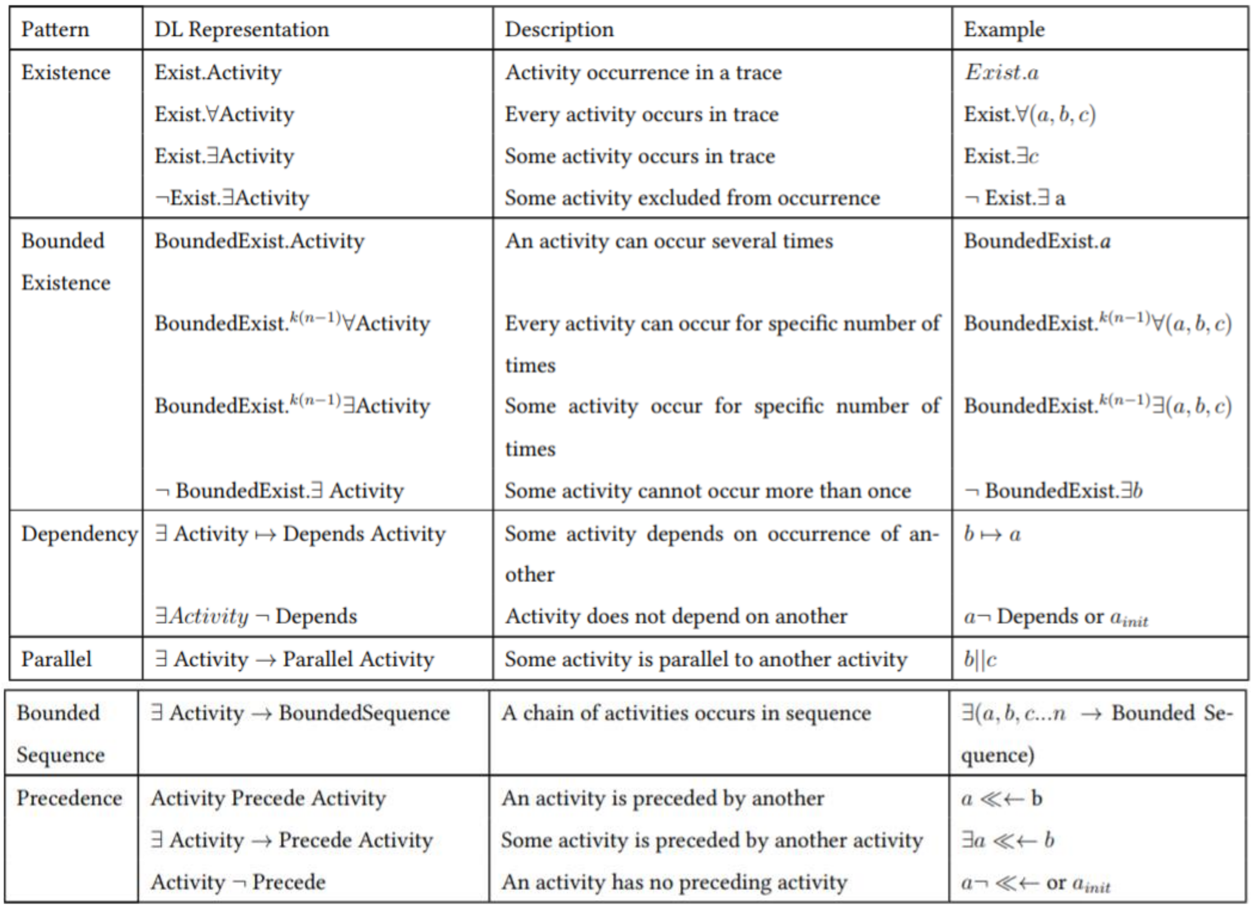
In this section, the presentation of constraints is categorised according to DL unary, binary and composite predicates. Unary predicates represent atomic constraints while binary and composite predicates represent combinations between constraints.

Unary Predicate Expressions for control flow Constraints

Control flow unary predicates are used to represent control flow-based constraints expressing ordering relations involving atomic activities or tasks. To fulfil the ordering relations, LTL operators and quantifiers are used for the purpose. The expressions are presented in **Table 7.** Activity combinations are required to express relations between one or more activities. Such relations are represented by forming combinations between constraints using combinations of predicates known as binary predicates.

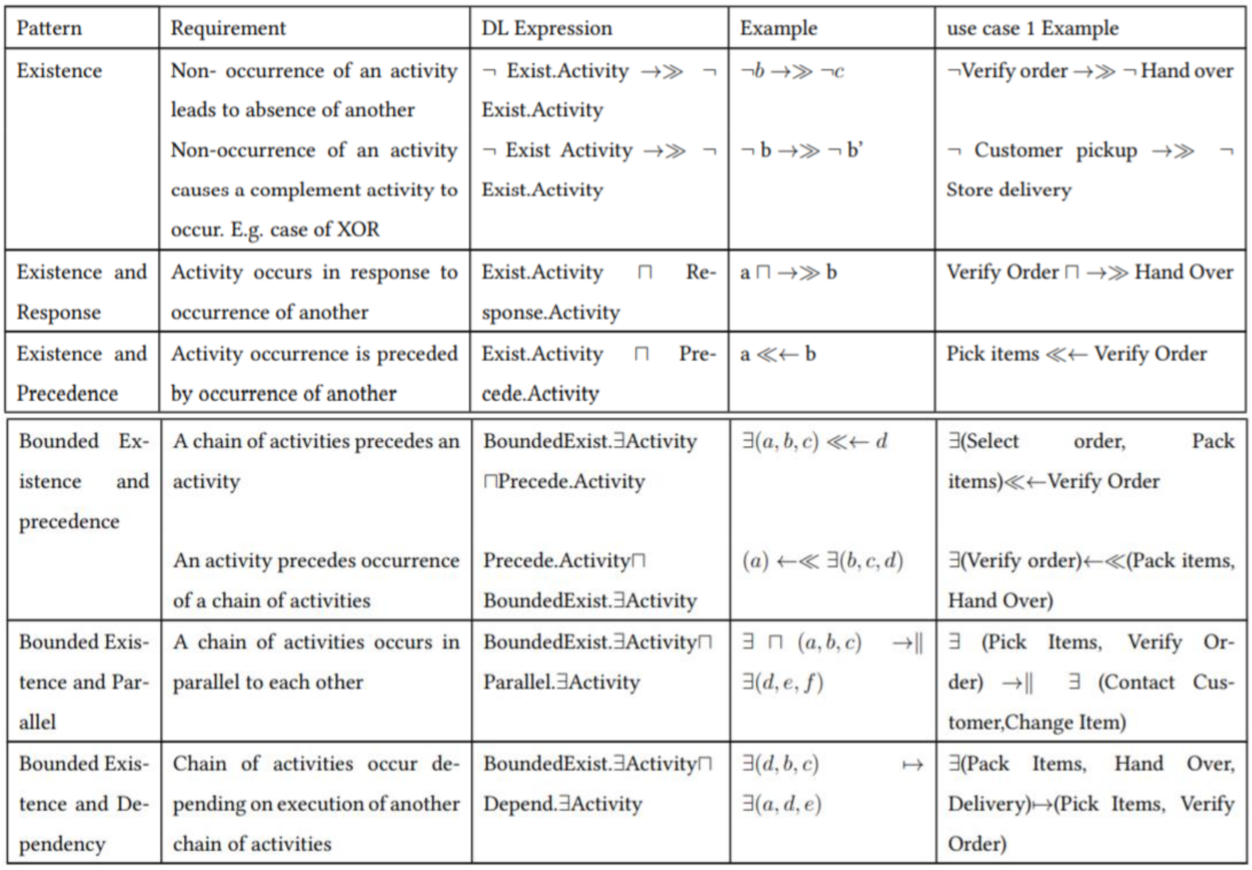
Binary Predicate Expressions for Control Flow Constraints

Two additional logical symbols are composed to achieve purposeful and meaningful expressions. These are: $\ll\leftarrow$ to represent 'precede', $\rightarrow\gg$ to represent 'leads to', $\parallel$ to represent parallel, and $\mapsto$ to represent dependence. **Table 8** presents binary expressions specifying constraints defined with examples from use case 1.



Pattern	DL Representation	Description	Example
Existence	Exist.Activity	Activity occurrence in a trace	$Exist.a$
	Exist.∀Activity	Every activity occurs in trace	Exist.$\forall(a,b,c)$
	Exist.∃Activity	Some activity occurs in trace	Exist.$\exists c$
	¬Exist.∃Activity	Some activity excluded from occurrence	¬ Exist.∃ a
Bounded Existence	BoundedExist.Activity	An activity can occur several times	BoundedExist.a
	BoundedExist.$^{k(n-1)}$∀Activity	Every activity can occur for specific number of times	BoundedExist.$^{k(n-1)}\forall(a,b,c)$
	BoundedExist.$^{k(n-1)}$∃Activity	Some activity occur for specific number of times	BoundedExist.$^{k(n-1)}\exists(a,b,c)$
	¬ BoundedExist.∃ Activity	Some activity cannot occur more than once	¬ BoundedExist.$\exists b$
Dependency	∃ Activity $\mapsto$ Depends Activity	Some activity depends on occurrence of another	$b \mapsto a$
	$\exists Activity$ ¬ Depends	Activity does not depend on another	a¬ Depends or a_{init}
Parallel	∃ Activity → Parallel Activity	Some activity is parallel to another activity	$b\|\|c$
Bounded Sequence	∃ Activity → BoundedSequence	A chain of activities occurs in sequence	$\exists(a,b,c...n$ $\rightarrow$ Bounded Sequence)
Precedence	Activity Precede Activity	An activity is preceded by another	$a \ll\leftarrow$ b
	∃ Activity → Precede Activity	Some activity is preceded by another activity	$\exists a \ll\leftarrow b$
	Activity ¬ Precede	An activity has no preceding activity	a¬ $\ll\leftarrow$ or a_{init}

Table 7. Unary Expressions for Control Constraints



Pattern	Requirement	DL Expression	Example	use case 1 Example
Existence	Non- occurrence of an activity leads to absence of another	¬ Exist.Activity $\rightarrow\gg$ ¬ Exist.Activity	$\neg b \rightarrow\gg \neg c$	¬Verify order $\rightarrow\gg$ ¬ Hand over
	Non-occurrence of an activity causes a complement activity to occur. E.g. case of XOR	¬ Exist Activity $\rightarrow\gg$ ¬ Exist.Activity	¬ b $\rightarrow\gg$ ¬ b'	¬ Customer pickup $\rightarrow\gg$ ¬ Store delivery
Existence and Response	Activity occurs in response to occurrence of another	Exist.Activity ⊓ Response.Activity	a ⊓ $\rightarrow\gg$ b	Verify Order ⊓ $\rightarrow\gg$ Hand Over
Existence and Precedence	Activity occurrence is preceded by occurrence of another	Exist.Activity ⊓ Precede.Activity	a $\ll\leftarrow$ b	Pick items $\ll\leftarrow$ Verify Order
Bounded Existence and precedence	A chain of activities precedes an activity	BoundedExist.∃Activity ⊓Precede.Activity	$\exists(a,b,c) \ll\leftarrow d$	∃(Select order, Pack items)$\ll\leftarrow$Verify Order
	An activity precedes occurrence of a chain of activities	Precede.Activity⊓ BoundedExist.∃Activity	$(a) \leftarrow\ll \exists(b,c,d)$	∃(Verify order)$\leftarrow\ll$(Pack items, Hand Over)
Bounded Existence and Parallel	A chain of activities occurs in parallel to each other	BoundedExist.∃Activity⊓ Parallel.∃Activity	∃ ⊓ (a,b,c) $\rightarrow\|\|$ $\exists(d,e,f)$	∃ (Pick Items, Verify Order) $\rightarrow\|\|$ ∃ (Contact Customer,Change Item)
Bounded Existence and Dependency	Chain of activities occur depending on execution of another chain of activities	BoundedExist.∃Activity⊓ Depend.∃Activity	$\exists(d,b,c)$ $\mapsto$ $\exists(a,d,e)$	∃(Pack Items, Hand Over, Delivery)$\mapsto$(Pick Items, Verify Order)

Table 8. Example Binary Expressions for Control Flow Constraints

4.3.3. Resource Constraints Expressions

For simplicity and convenience, abbreviations are adopted for use in expressions representing resource constraints as follows: Available - Avail, Segregation of duty - SoD, binding of duty - BoD and Delegation - Del.

Unary Resource Constraint Expressions

Unary expressions representing resource constraints and their implications using description logic are as follows: •

- Resource.SoD – Resources constrained with SoD constraint
- Resource.BoD – Resource is constrained with BoD constraint
- Resource.Del – A resource that can be delegated

Binary Resource Constraint Expressions

Binary combinations between resource constraints is possible under guiding principle that no combination between BoD and SoD for same activity executions at the same time. This comprises the access control restrictions, i.e. a resource cannot be constrained as BoD and SoD at the same time of allocation to an activity. This could otherwise result into deadlocks. Similarly, a resource cannot be available and unavailable at the same time. These restrictions must be observed at design time and verified to prevent violations that result into noncompliance or deadlocks. Some of the realistic constraint combinations are:

- Resource.Avail ∩ SoD – Resource available for assignment as SoD.
- Resource.SoD ∩Del – Resources constrained with SoD and can be delegated.
- Resource.BoD ∩Del – Resource is constrained for BoD but can be delegated.
- Resource.Avail ∩ Validity [time] – Resource's availability is valid for a specific time.

4.3.4. Data Constraint Express

Unary Data Constraint Expressions

The section presents data constraints expressions based on Unary predicates using description language:

- Data.visible – Data items visible for each activity.
- Data.¬ visible – Data items not visible
- Data.interactive – Data items that can be interacted with.
- Data.valid - Valid data items
- Data.available - Available data items
- Data.¬available – Data items unavailable
- Data.accessible - All accessible data items
- Data.¬accessible – Data items inaccessible.
- Data.Privacy – Data items classified as private data
- Data.2-3-4WM – Data that requires matching to enable execution.
- Data.Authentication - Data items that require authentication.

Data constraints restrict the creation and access of the data by activity tasks and resources (roles and applications) over time.

4.4. Composite Predicate Constraint Expressions

This section presents composite predicate combinations involving all constraint categories to fulfil compliance requirements. They include the following:

4.4.1. Expressions Between Control Flow and Resource Constraints

Combination between control flow and resource constraints expresses conditions restricting assignment of resource actors to activities. The expressions specify activity behaviour in relation to their actors. The DL based expressions are represented as follows:

- Exists.Activity → Resource.Avail – The occurrence of an activity is assigned to an available resource actor. For example:

  $Ho \rightarrow Resource.[Avail]$ is a valid expression assigning any actor that will be available to execute the activity. Such activities assigned to any available actor are non-critical or, they are already within the category of authorized actors.

- Exists.Activity → Resource.SoD – The occurrence of the activity is assigned to a resource actor constrained by separation of duty. For example:

  $Vo \rightarrow Resource.[Avail]$ is a valid expression assigning the actor or role Verifier constrained by SoD to verify order activity.

- $Exists.Activity1 \sqcap Activity2 \ \rightarrow Resource.BoD$ – The occurrence of the activity is assigned to a resource actor constrained by binding of duty. For example:

  $So \sqcap Pit \rightarrow Picker.[BoD]$ is a valid expression assigning the actor role Picker constrained by BoD to execute Select order and Pick items activities. This implies that the actor executes both activities.

- $Exists.Activity \ \rightarrow Resource1.Del: Resource2$ – The occurrence of the activity is assigned to a resource actor that can delegate to another actor. For example:

  $Po \rightarrow Packer.[Del]: Picker$ is a valid expression assigning the actor of role Packer who can delegate to actor Picker. Therefore a given activity Ho can be delegated to several actors such that $Ho \rightarrow \sqcup[R_1, R_2, \dots R_n]$ implies that Hand over order is assigned to R_1, R_2 and R_3.

- $Exists.\forall Activity \ \rightarrow Resource.SoD$ – The occurrence of a set of activities is assigned to a set of resource actors constrained by separation of duty. For example:

  $Vo, Mo, CallCustomer \rightarrow Verifier, Supervisor, DutyManager.[SoD]$ is a vali expression assigning resource actors of role Verifier, Supervisor and Duty Manager constrained by SoD to a set of activities: Verify order, modify order and call customer.

- $BoundedExists^{(n+1)=k}.Activity \rightarrow Resource.$ Several occurrences of an activity are assigned to the same resource. For example:

  $BoundedExists^{(n+1)=k}.PickItems \rightarrow Picker.$ All the number of times within an instance of pick items are executed by the same actor with picker role.

- $BoundedExists^{(n+1)=k}.Activity \rightarrow Resource[SoD]$ – The number of times an activity can occur it is executed by a different resource. For example:

  $BoundedExists^{(n+1)=k}.VerifyOrder \rightarrow (\text{Verifier1} \sqcap \text{Verifier2}).[\text{SoD}].$ All the number of times the event instance of verify order is executed by a different actor (verifier1 or verifier2) of the assigned role (Verifier).

- $BoundedExists^{(n+1)=k}\exists Activity \rightarrow\sqcap \exists Resources[BoD]$ – Several occurrences of activities are assigned to the same resource constrained through binding of duty for all occurrences.
- $BoundedExists^{(n+1)=k}\exists Activity \rightarrow\sqcap \exists Resources[SoD]$ – Several occurrences of activities are assigned to different resources, constrained as separation of duty for all their event instances.
- $BoundedSequence^{k}\exists Activity \rightarrow \exists Resources.[SoD]$ – Activities occur as a chain for a number of times are assigned to different resources constrained by separation of duty.
- $BoundedSequence^{k}\exists Activity \rightarrow \exists Resources.[BoD]$ – - Several activities occur as a chain for a number of times are assigned to a resource actor constrained by binding of duty.

Other expressions for control flow and resource constraints can be defined and specified following the same syntax and semantics. The above expressions are for illustration purposes.

4.4.2. Expressions for Control Flow, Resource, Data and Temporal Constraints

This section presents predicate combinations for all constraint categories. The combinations represent means for complete constraint specifications that is close to natural language. This way, non-expert end users can extract compliance requirements from source policy and regulatory documents and represent them as constraints to be complied with by the business processes.

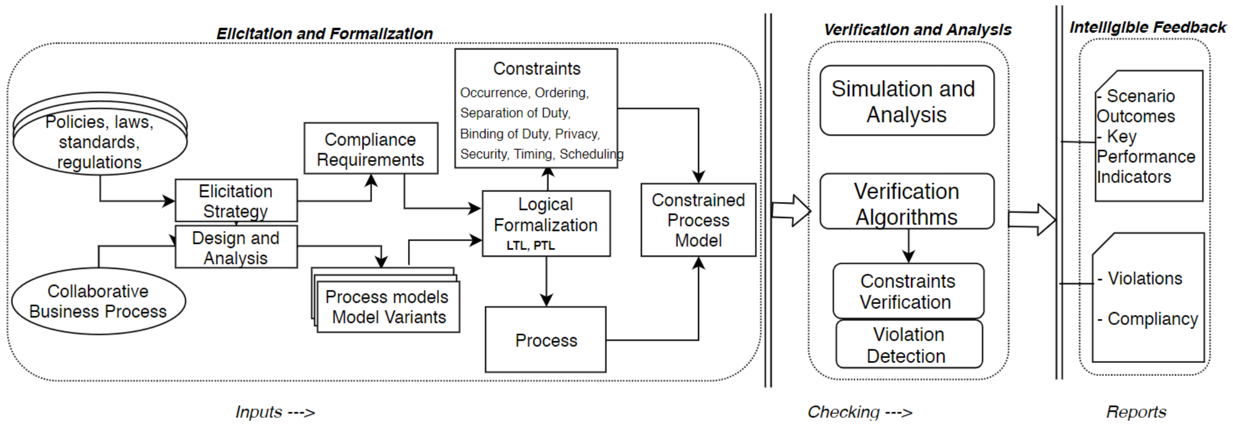
- $Exists.Activity \rightarrow Resource.[SoD] \sqcap Data.[available] \sqcap Time.[Duration]$ – The expression specifies that an activity is assigned to some resource constrained by separation of duty, and data access as available for a specific duration.
- $Exists.Activity \rightarrow Resource.[BoD] \sqcap Data.[Private] \sqcap Time.[Duration]$ – The expression specifies that an activity will occur, assigned to a resource constrained as binding of duty, with data constrained with privacy for a duration of time.
- $BoundedExists^{(n+1)=k}.Activity \rightarrow Resource.[BoD] \sqcap Data.[Authentication] \sqcap$ $Time.[Duration]$ – The expression specifies an activity that will occur several times, with each time to be executed by a resource constrained by binding of duty, and to access data by authentication for a specific duration of time.
- $BoundedSequence^{k}\exists Activity \rightarrow Resource.[Del] \sqcap Data.[Authentication] \sqcap Time.[Duration]$ – A set of activities to be executed for a number of times in sequence are assigned to resources which can delegate and share execution rights to other resources.

5. Compliance Verification Approach (Definition and Algorithms)

Figure 5 presents an overall compliance verification approach showing three main steps. ***The first step*** is compliance constraints specification, in this step the relevant rules and policies are extracted from source documents and compiled into a set of compliance requirements, defined to guide process behaviour. The set includes all requirements relevant for an organization's business processes to comply with as sourced from all policies, contractual obligations and external regulations. To support reasoning, model logic is used to translate the requirements into formal compliance constraints. In this case both Description logic and linear temporal logic are used.

The second step is compliance verification; here, the business process model is verified for its compliance with formalized constraints. The goal is to check and ensure that the business process conforms to the required policies and regulations. Relatedly, in this step simulation analysis is used to illustrate the impact of change and variation in policy and regulations over the business process.

The Third step is the outcome of the verification forms the feedback reports displayed for users about compliancy or violation of the constraints. Outcome from simulation analysis shows the scenario reports and key performance indicators.



Elicitation and Formalization
Policies, laws, standards, regulations
Elicitation Strategy
Design and Analysis
Collaborative Business Process
Compliance Requirements
Constraints
Occurrence, Ordering, Separation of Duty, Binding of Duty, Privacy, Security, Timing, Scheduling
Logical Formalization
LTL, PTL
Process models Model Variants
Process
Constrained Process Model
Verification and Analysis
Simulation and Analysis
Verification Algorithms
Constraints Verification
Violation Detection
Intelligible Feedback
- Scenario Outcomes
- Key Performance Indicators
- Violations
- Compliancy
Inputs --->
Checking --->
Reports

Figure 5: Overall Compliance Verification Approach

5.1. Categories of Constraint Verification '

The verification component of the compliance approach is formed of 2 types of checking:

The Simulation component: Simulation is undertaken to generate traces to facilitate analysis and verification. The analysis involves predictive performance assessment of the business process based on variations in policy and regulations. Differing scenarios are generated and outcomes are analysed to support informed decision making.

The verification algorithm component: 'This component is formed of algorithms that identify and detect compliance constraints violations. Various algorithms are composed for categorical constraint verification applicable in different ways, e.g. if a policy changes, users may want to check for compliance of existing processes with the changed policy. This way, only the relevant algorithm applies. An alternative is using the overall verification algorithm that combines all categories. Procedurally a business process is checked for compliance with all relevant constraints. 'this applies to new business process or those that have been modified significantly. In either case, the checking procedure in Figure 3 is followed. A business process is checked by detecting compliancy or violations to required behaviour expressed as constraints. Further, details of the checking are described in the algorithms presented in subsequent sections.

Figure 6 illustrates the compliancy verification procedure. 'The existing or new business processes are checked for conformance with defined constraints. If the process model is compliant, feedback is given, otherwise detection of non-compliant behaviour proceeds. Where the algorithms detect non-compliant behaviour, specific or general feedback is given about the violations. To enable independent constraint checking, algorithms are composed according to same categories to permit constraint specific checking without need to follow a step wise procedure every time. The following section presents the algorithms according to their categories.

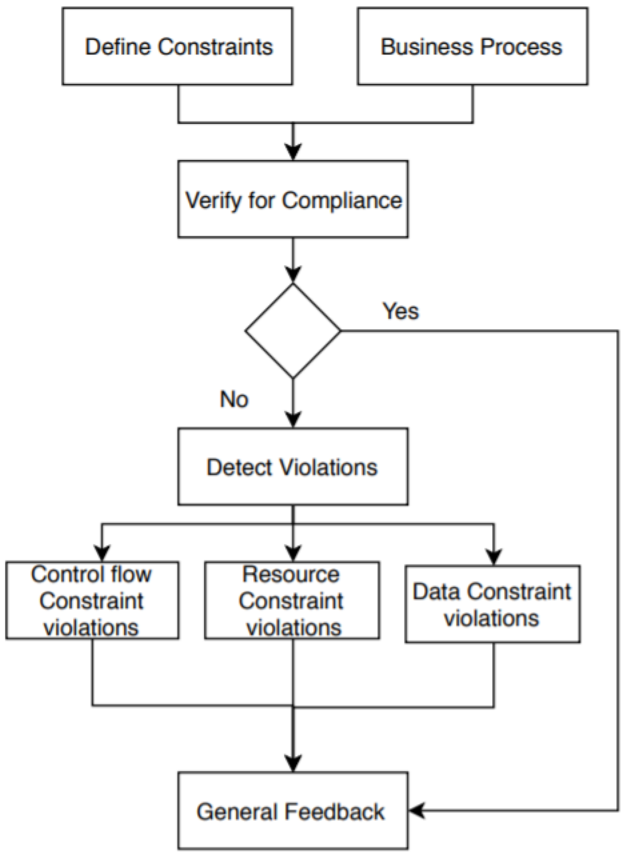


Figure 6: Compliance verification procedure

5.2. Control Flow Verification

The compliance verification algorithms that will be introduced later facilitate business process designers to check for the well-connectedness of the models to ensure that there are no errors like; 1) deadlocks, 2) improper termination, and 3) live locks. A well-connected model facilitates checking for other system model properties like safety and liveness. Safety is a notion that nothing will go wrong in the model while liveness principle states that something good will happen. This section presents the definitions and specifications for the functions that are used by the verification algorithms. The definition follows the constraints categories.

5.2.1. Control Flow Verification Requirements

Connectedness of the process model: Verification of how a process model is well connected is based on the modelling constructs like Sequence, AND, XOR and OR. It is important for the model to be well- formed from the design point of view even before other properties can be checked. This way, if a model’s structural requirements are satisfied, then its soundness is consequently achieved (van der Aalst, 1997, 2002; Wynn *et al.*, 2009). At this level, verification targets to check how structurally well formed a model is in terms of sequence, parallelism, exclusive and inclusive choice constructs. In this section the structural requirements are defined and later we show how to verify for their conformance.

- Sequence: checking sequential connection between model objects. A valid sequence is given by:

$Sequence = \sigma_i(a_1 + \cdots + a_n) \in P$i

A sequence is a trace of activities from the initial to the n^th activity in a process instance satisfying a predefined order.

- Parallelism: checking connection between objects representing two or more tasks executed simultaneously and the possibility to converge at another object.

  $AND = \sigma_i\big((a_1 - a_2) \wedge (a_1 - a_3)\big) \in Pi$

  For a given trace in a process instance, any two interleaving tasks with no partial order relation conform to execution constraints if both tasks execute as per the constraint requirement.

- Exclusive choice: checking connection between objects representing disjoint tasks where one of them should execute.

  $XOR = \sigma_i\big((a_1 - a_2) \vee (a_1 - a_3)\big) \in Pi$

  For a given trace in process instance, any two disjoint tasks with no partial order relation conform to execution constraints if either of the tasks executes as per the constraint requirements.

- Inclusive choice: checking for connection between objects representing tasks where one or more alternative tasks can execute from a set of alternative paths.
- $OR = \sigma_i\big((a_1 - a_2) \wedge (a_1 - a_3) \wedge (a_1' - a_3')\big) \in Pi$

  For a given trace in a process instance, any two joint tasks with no partial order relation conform to execution constraints if one or of the tasks executes as per the constraint requirements.

5.2.2. Specification of Control Flow Constraints

Control flow constraints include among others, existence and bounded existence, dependency, bounded sequence and precedence. Compliance to these constraints is verified in relation to temporal constraints to ensure that task ordering and occurrence follow time requirements. To facilitate the checking, we make the following definitions:

Specification for Existence (and Bounded Existence)

Existence constraint restricts an activity to occur in a specific order or time within a trace of a process instance. It also specifies ordering relations where specific activity events must start (einit) or end (eend) an instance. ‘This way, the validity of an instance can be checked. To this effect definition 5.2 refers.

Definition 5.2.1 Existence (and Bounded Existence)

1. Existence for process instance validity.

   $Check.Exist: (e.ac = init) \cap (e.ac = end) \in \sigma$ Where: $e.ac$ is the event of an activity. The expression specifies a function to check initial and end activity events in a trace.
2. Existence of an activity within a process instance checked in reference to the control structures
3. If $(e.ac = AND)$ Return $\uplus\big((a_1, a_2) \sqcap (a_1, a_3)\big)$
4. If $(e.ac = XOR)$ Return $\uplus\big((a_1, a_2) \sqcup (a_1, a_3)\big)$
5. If $(e.ac = OR)$ Return $\uplus\big((a_1, a_2) \sqcap (a_1, a_3) \sqcap (a_1, a_4)\big)$

Application of the function

To illustrate the application of the function above, data in Table 6 is used to check the constraint requirements.

```
for each $\sigma \in Pi$ do
    $Check.Exists: (e.ac = init) \sqcap (e.ac = end)$
end for
Return
$e.ac = init \notin seen$                /*Initial event is not in 'seen' events of the instance */
$e.ac = end \notin seen$                 /* End event is in 'seen' events of the process instance. */
```

Using data populated in Table 6 with events, activities and process instances, we show the application of existence constraint specification and checking for its compliance or violation. Figure 7.4 shows resultant state graphs generated from the constraint checking of existence and bounded existence for all structural constructs (sequence, AND, exclusive and inclusive choices). The following verification requirements are addressed:

Requirement 1: All process instances start and end with activities a and z respectively.

Requirement 2: Between activities a and z, a set of other activities execute as part of the process instance.

Instances	Pi1				Pi2				Pi3						Pi4				Pi5		
Events	e1	e2	e3	e4	e5	e6	e7	e8	e9	e10	e11	e12	e13	e14	e15	e16	e17	e18	e19	e20	E21
Activities	a	b	e	z	a	e	c	z	a	b	f	g	h	Z	a	i	m	z	a	z	m
time	2	4	3	5	2	3	6	5	2	4	6	4	8	4	3	4	3	5	3	3	3

Requirements 1 and 2 in the section above can be checked in the following way using the specified expressions.

```
for $\sigma \in Pi$ do $Check.Exist: (e.ac = init) \cap (e.ac = end)$
    Return
    $init = a\forall Pi$          /*Returns activity a as initial activity for all process instances*/
end for
```

Based on the expressions, it follows that activity a is the initial activity for each process instance, so is activity z for end activity in each process instance.   In terms of soundness, it shows compliance to termination is achieved by the possibility that each instance can start at a and end with z. However, the checking is not complete until we check for any possible violations of the behaviour.

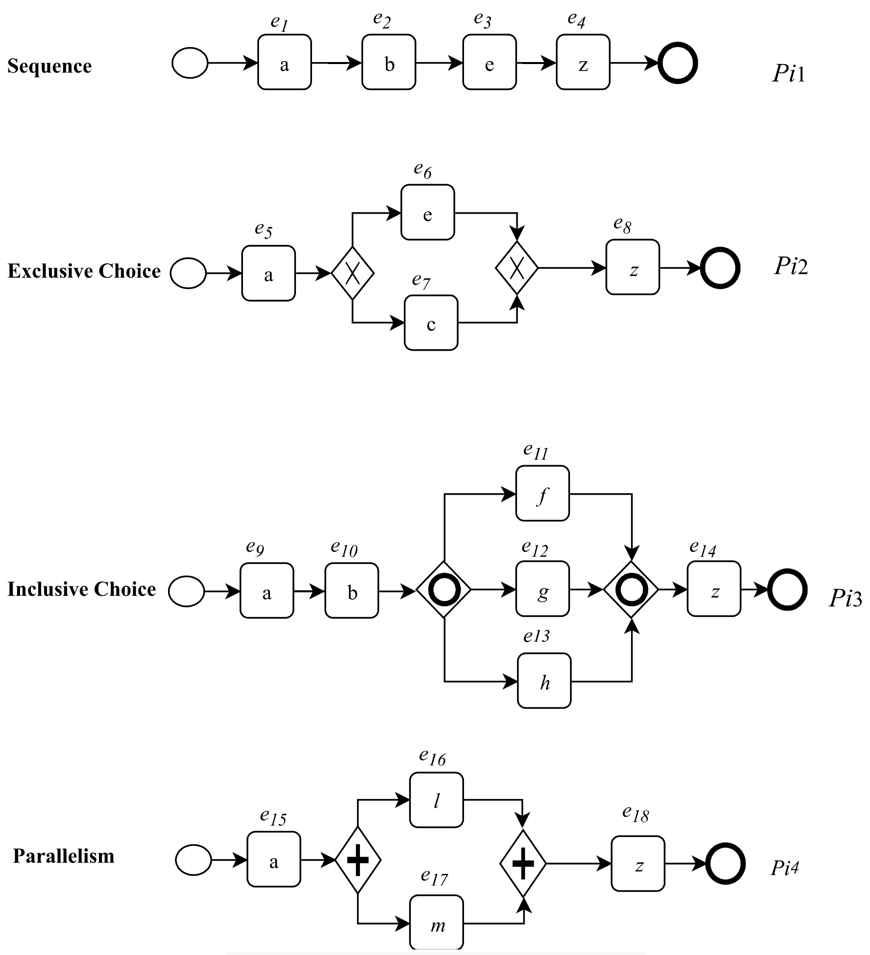


Figure 7: Resultant State Graphs

Constraint Satisfaction Checking

We adopt to predicate functions for representing constraint satisfaction or violation.

- seen - Represents running activity events. If it is True that an activity event or set of activity events is in seen (e.g. $ac \in seen$), then the constraint is satisfied ($True \vDash C$). Otherwise it is violated ($True \nvDash C$).
- finished - Represents executed activity. If it is True that an activity event or set of activity events is in finished (e.g. $ac \in finished$), then the constraint is satisfied ($True \vDash C$). Otherwise it is violated ($True \nvDash C$) events.

Detecting violation to existence constraint

Violations to existence constraint are detected by checking for instances in which activities *a* and *z* are not initial and end activities respectively, and where the initial time assignments are not observed for all events. Circumstances leading to violation are checked from:

1. Process instances where activity *a* is not the initial activity in a set of process executions, i.e.

   $a \in seen$

   From Table 6, it shows that events (e15, 3, P i4) partially satisfy the constraint since *a* is the initial activity for all instances. However, in terms of the temporal requirement the activity executes for longer time than scheduled, i.e. 3 units of time instead of 2 units.

2. Process instances where activity z is not the end activity in all process executions,

   i.e. $z \notin finished$

From Table 6, it shows that trace (e20, 5, Pi5) involves constraint violating event. Activity *z* is not the end activity for the constraint. There is a variance in execution duration where less than time is used 3 units are used compared to what was scheduled 5 units). This saves time as opposed to being a violation.

Specifications for Precedence and Dependence Constraints Verification

Precedence and dependence constraints are verified for activities whose existence has been confirmed. To verify that activity *b* is preceded by *a* and that the occurrence of *b* determines occurrence or non-occurrence of another activity **c,** we check for occurrence of *b* and return its preceding activity as well as the activity that occurs after its execution as its dependent activity, in other words activity *c* occurrence depends on activity b. 'The constraint is specified as the expression below:

Definition 5.2.2. Precedence and Dependence

$Check.Precede = (a \ll\leftarrow b)$ *\/* checks for precedence of a over b*\/*

$Check.Depend = (c \mapsto b)$ *\/*checks for dependence c on b*\/*

The expressions define activity *a* as a preceding activity to *b*, while occurrence of activity *c* is dependent on b such that *c* occurs if and only if *b* has occurred (Xu, 2003; Xu and Jeusfeld, 2003). The definition is used to specify constraint checking expression for the different control structures which are afterwards used in the algorithms. The checking involves:

1. Checking if an activity has occurred in the trace $e.ac \in \sigma$.

   Return error if $e.ac \in \sigma$. stop checking.

2. Check for precedence and dependence constraints and returns outcome based on the routing constructs:

```
**While** $e.ac \in \sigma$ **do**
    $((e.ac = a) \to Precedes(e.ac = b)) \wedge ((e.ac = c) \to Depends(e.ac = b)): (\exists c) \leftrightarrow (\exists b)$
**Return** $(e(i <= j)) \in Pi$    /* Returns events satisfying or violating the constraints e.g. c
                                   occurs if and only if b occurs. Otherwise it is a violation*/
```

   i. If AND /*output based on AND construct */

$$\begin{Bmatrix} \cap_{i\leq j}^{e} e.ac(a.Precedes(b)) \in seen = True \vDash C \\ \cap_{i\leq j}^{e} e.ac(a.Precedes(b)) \notin seen = False \nvDash C \end{Bmatrix}$$

   While verifying precedence constraint for activities based on AND construct, the checking returns a false if there are no seen events where activity a precedes activity b.

$$\begin{Bmatrix} \cap_{i\leq j}^{e} e.ac(c.Depends(b)) \in seen = True \vDash C \\ \cap_{i\leq j}^{e} e.ac(c.Depends(b)) \notin seen = False \nvDash C \end{Bmatrix}$$

   While verifying dependence constraint for activities based on AND construct, the checking returns a false if there are no seen events in which activity c depends on b

   ii. If XOR construct */output based on XOR construct

$$\begin{Bmatrix} \cup_{i\leq j}^{e} e.ac(a.Precedes(b)) \vee (a.Precedes(b')) \in seen = True \vDash C \\ \cup_{i\leq j}^{e} e.ac(a.Precedes(b)) \vee (a.Precedes(b')) \notin seen = False \nvDash C \end{Bmatrix}$$

   Outcome for events satisfying or violating the precedence constraint on disjoint activities *b* and *b*' over activity *a*. A violation occurs when activity a is not seen among activities preceding activity b for all instances

$$\begin{Bmatrix} \cup_{i\leq j}^{e} e.ac\big(c.Depends(b)\big) \vee \big(c'.Depends(b)\big) \in seen = True \vDash C \\ \cup_{i\leq j}^{e} e.ac\big(c.Depends(b)\big) \vee \big(c'.Depends(b)\big) \notin seen = False \nvDash C \end{Bmatrix}$$

Set of events satisfying or violating the dependence constraint for disjoint activities c and c' over activity b. A violation occurs when activity b is not in seen activities where activities c and c' are seen among activities for the process instances.

iii. If OR /*Outcome based on OR construct*/

$$\begin{Bmatrix} \cup_{e+1}^{e} e.ac\big(a.Precedes(b)\big) \vee \big(a.Precedes(b'^{\mathrm{n}})\big) \in seen = True \vDash C \\ \cup_{e+1}^{e} e.ac\big(a.Precedes(b)\big) \vee \big(a.Precedes(b'^{\mathrm{n}})\big) \notin seen = False \nvDash C \end{Bmatrix}$$

The occurrence of activity b is preceded by activity a where more than one alternative paths are permissible. If events of activity a are in seen and finished, then the precedence constraint is satisfied. Otherwise it is violated.

$$\begin{Bmatrix} \cup_{e+1}^{e} e.ac\big(a.Depends(b'^{n})\big) \vee \big(a.Depends(b'^{\mathrm{n}})\big) \in seen = True \vDash C \\ \cup_{e+1}^{e} e.ac\big(c.Depends(b)\big) \vee \big(c'^{n}.Depends(\mathrm{b})\big) \notin seen = False \nvDash C \end{Bmatrix}$$

The occurrence of activity b is preceded by activity a. If events of activity a are in seen and finished occurring before activity a, then the dependence constraint between a and b for all alternative paths is satisfied. Otherwise it is violated.

iv. If Sequence: constraint checking based on sequence construct is checked in the same way as specified expressions illustrated above.

Definition 5.2.3. Other control flow constraints

The illustration involved the definition and specification of existence, bounded existence, precedence and dependence constraints. However, other control flow constraints like Response, bounded response inter alia can be extended into definitions and specifications in the same way as illustrated in sections above. For time and space limitations not all control flow constraints are specified. After the definitions and specification of constraints and checking functions, control flow compliance checking algorithms are composed.

5.2.3. Control Flow Verification Algorithm

Based on the above discussions, specifications and function definitions, a set of control flow based algorithms are composed to check compliance of the business process with control flow constraints. To make the algorithms self-contained and independent the definitions below are used for all algorithms. The general assumption is that events are ordered in a total order over time.

Predicate Functions

- Business process: $= BP$
- Process Instances: $Pi = \{\sigma i \ldots, \sigma n\}$
- Trace (σ): Logical activity events.
- Events in a trace = started, seen, € Finished where;

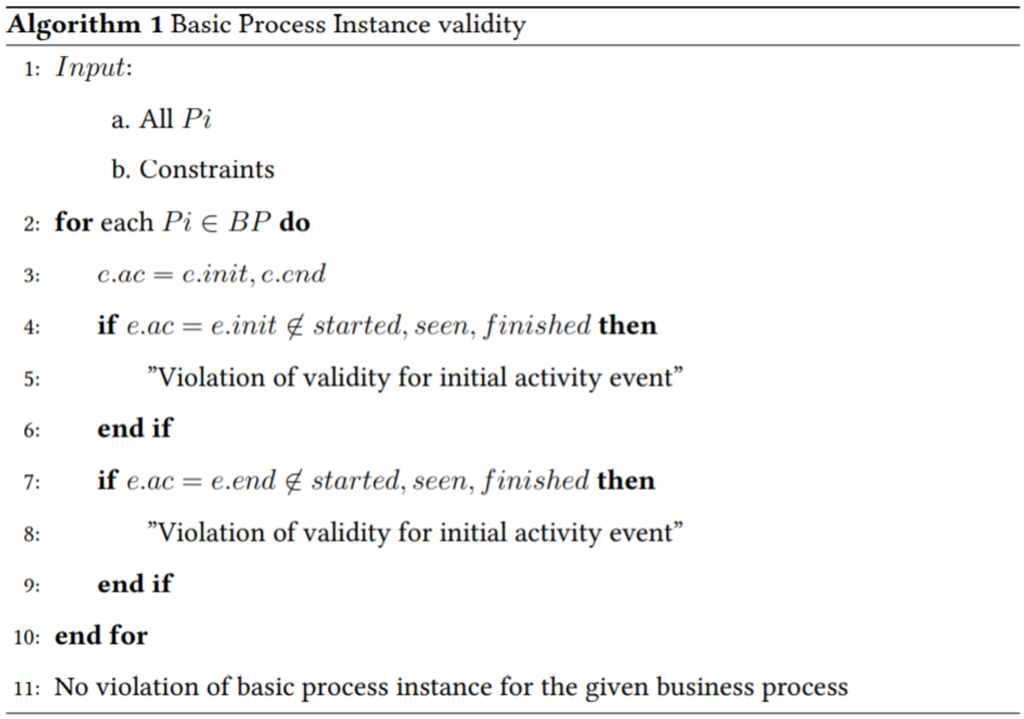
  – started = {} − Set of started activity events.

  – seen = {} − Set of seen or running activity events.

  – finished = {} − Set of finished activity events.
- e.ac: Activity

Events Verifying for Basic Process Instance Validity

Sub-**algorithm 1** checks for the basic validity of the model based on activity events that start and end a process instance. ‘e algorithm checks for activity events designated to start or end a process instance. If start events are not in a set of ‘started’ events (e.ac ∉ started), it implies the activity has not started. If it is not in ’seen’ activities (e.ac ∉ seen), or ’finished’ (e.ac ∉ finished), it implies that the activity is not in execution or not completed. The same principle applies for the end activity events. In this case a violation is reported for activities not started, not in seen and not in finished.



Algorithm 1 Basic Process Instance validity

```
1: $Input$:
       a. All $Pi$
       b. Constraints
2: for each $Pi \in BP$ do
3:     $e.ac = e.init, e.end$
4:     if $e.ac = e.init \notin started, seen, finished$ then
5:         ”Violation of validity for initial activity event”
6:     end if
7:     if $e.ac = e.end \notin started, seen, finished$ then
8:         ”Violation of validity for initial activity event”
9:     end if
10: end for
11: No violation of basic process instance for the given business process
```

Verifying for Compliance with Existence constraint

The existence constraint refers to constraints that restrict the occurrence behaviour of an activity. The algorithm verifies for occurrence of activity events in a process instance as per required behaviour specified by the policies governing operations. The events are fully ordered by time. It is intended to address the following verification requirements;

Requirement 2.1: Check out activities scheduled to occur but never execute.

Requirement 2.2: Detect deadlocks by checking activities that start but never complete execution.

Based on **algorithm 2**, violation of the existence constraint is detected if any of the event activity states is not among the events that are started, executing or completed within the seen and finished event sets.

Algorithm 2 Existence Constraint Checking

```
1: $Input$ :
        a. All $Pi$
        b. Constraints
2: for each $Pi \in BP$ do
3:     $e.ac.State \in Started, Executed, Completed$
4:     if $e.ac.State \notin Started, Executed, Completed$ then
5:         ”Violation: Existence constraint violated. Activity never occured”
6:     end if
7: end for
8: No violation of existence constraint for the given business process
```

Verifying for Compliance with Precedence constraint

Precedence constraints restrict the ordering relations between activities based on occurrence of a previous activity. In collaborative business processes characterized by multi-party executions, checking the precedence of activities benefits transparency in partner responsibility by knowing which activities must occur before others and who should execute them. In case of deadlocks, it is possible to point to the source of the problem. To facilitate verification of compliance with precedence constraints for activities, **algorithm 3** is composed and presented addressing the following requirements:

Requirement 3.1: Detect activities that are potential sources of precedence violation.

Requirement 3.2: Use compliant behaviour to determine any likely violations based on the routing constructs.

The algorithm checks precedence condition activity event over an action event. Violation occurs where the condition does not lead to the action or where the action occurs without the condition activity. For example, activity a_1 is the precedence condition for occurrence of activity a_2. The occurrence of a_2 before occurrence of a_1 is a precedence constraint violation that **algorithm 3** identifies.

Algorithm 3 Precedence Constraint Checking

```
1: $Input$ :
       a. All $Pi$
       b. Constraints (Precedence)
2: for $Pi \in BP$ with Precedence constraints C do
3:     $Prec = e.ac.Condition \Rightarrow e.ac.Action$
4:     if $(Prec \notin seen, finished)$ then
5:         Violation ("Precedence constraint violated")
6:     end if
7: end for
8: No violation of Precedence event in the business process
```

Verifying for Compliance with Response constraint

Response constraint restricts execution of activities based on evaluation of a condition on the current activity. The activity will then execute in response to the outcome of that condition e.g. If a cheque is approved, then it can be issued. Issue cheque is a response activity from approve cheque. Execution issues arise if the condition is not evaluated or evaluates falsely leading to deadlocks or live locks. **Algorithm 4** in this section checks for compliancy with response constraint over a set of activities. The following verification requirements are addressed:

Requirement 4.1: Detect activities likely to lead to response-based violations.

Requirement 4.2: Detect deadlocks resulting from non-responsive activities.

Algorithm 4 checks for Response constraint between activity events where an activity condition (e.ac.Condition) responds to an action activity event (e.ac.Action) where, occurrence of the action activity in the seen and finished events not as a response from the conditional activity event violates the response constraint.

Algorithm 4 Response Constraint Checking

```
1: $Input$ :
       a. All $Pi$
       b. Constraints
2: for all $Pi \in BP$ with Response constraints C do
3:     Response = e.ac.Action $\Rightarrow$ e.ac.Condition
4:     if ($Response \notin seen, finished$) then
5:         Violation:    "Response constraint violated"
6:     else if e.ac.Action $\Rightarrow$ e.ac.Condition $\neq$ Response $\in seen, finished$ then
7:         "Violation, condition activity occurred without induced action activity".
8:     end if
9: end for
10: No Violation of response constraint on the provided business process instances.
```

5.3. Resource Compliance Verification

Verification for compliance with resource constraints aims at checking for the fulfilment of the resource requirements by the business process such that no violations exist in its behaviour.

5.3.1. Specification of Resource Constraints

This section specifies the resource constraints as formal expressions and functions applicable in the resource verification algorithms to detect violations. The constraints are separation of duty, binding of duty and delegation.

- Separation of Duty (SoD): Requires two disjoint activities (a_1, a_2) to be executed by different resource actors (r_1, r_2). Such assignment is based on preliminary specification for actor (user) and task assignment. In light of the above, SoD specification for r_1, r_2 over (a_1, a_2)) is defined as:

  Definition 5.3.1. SoD

  $\nexists r_1 \in U: ((a_1, a_2), r_1)) \in RP$

  The assignment of SoD constraint serves as a guard preventing a single actor in a role from executing two disjoint activities. It follows therefore that there should not exist any assignment of an actor r_1 to execute both activities (a_1) and (a_2) in a user task assignment. The contrary is a constraint violation.

- Binding of Duty (BoD): BoD requires two tasks (a_1, a_2) to be executed by the same resource actor (r_1). BoD verification checks to ensure compliance to this requirement, the contrary of which is a violation. Following preliminary definitions above, specification for activities (a_1) and (a_2)) as BoD i.e. BoD (a_1, a_2) is given by the definition:

  Definition 5.3.2. BoD

  $r_1 \in RP: \forall\big((a_1, a_2), r_1\big) \in RP$

  For each actor assignment involving activities (a_1) and (a_2), one actor should be assigned for their execution. Contrary to the assignment is a constraint violation.

- Delegation: For tasks designated to specific resource actors, delegation enables sharing of execution rights with other actors. Two scenarios result where; the delegator shares and retains execution rights to the object or completely delegates and retains no execution rights to the delegate. Delegation is a practice in business operations to ensure business continuity. It also guards against activity dead locks that result from over constrained resources that create time lags and delays, or improper implementation of constraints like the four-eye principle.

Specification of the delegation constraint requires information about subjects (users who delegate and those delegated to), and objects. Therefore, given two (2) users r_1 and r_2 where r_1 delegates activity a to r_2, the expression below specifies the delegation constraint:

Definition 5.3.3. Delegation

$(a, r_1) \in UT|r_1 \rightarrow Delegate(a, r_2): (a, r_1 \wedge r_2)$

User (r_1) with rights to activity *a* delegates rights to user r_2 but retains execution rights such that both users are now assigned to activity a. $(a, r_1) \in UT|r_1 \rightarrow$ Delegate (a, r_2) Similarly, the above specification indicates that User (r_1) with rights to activity a delegates to (r_2) by passing on all the execution rights such that the delegator can no longer execute the activity.

5.3.2. Definitions for Resource Constraints

To facilitate the checking of compliancy to resource constraints, the following definitions are relevant. Given a trace $\sigma \in (a_1, a_2, a_3)$ and a set of two users' r_1 and c of instance Pi_1, the following functional definitions are employed by the algorithm during resource constraints compliance verification

```
While $\sigma \in (a_1, a_2, a_3), (r_1, r_2) = Pi_1$ do
  $Check.SoD = ((a_1, r_1) \wedge (a_2, r_2))$        /* checks compliance to user assignment over
                                                       activities $a_1$ and $a_2$ based on SoD constraint*/
  $Check.BoD = ((a_1, a_2), r_1)$                     /* checks compliance to actor assignment over
                                                       activities $a_1$ and $a_2$ based on SoD constraint */
  $Check.Delegate = (a, r_1 \wedge r_2)$              /* checks compliance to delegation constraint
                                                       for activity a between actors $r_1$ and $r_2$ */
```

Return is used to generate the outcome from compliance checking showing whether compliance or violation is achieved based on the different structural controls i.e. AND, Parallelism, OR and XOR.

5.3.3. Resource Compliance Verification Algorithms

The resource verification algorithms apply the specifications and definitions in previous section to check process behaviour. The previous definitions are applicable for algorithm 5:

Algorithm for SoD Constraint Verification

Verifying for this constraint involves checking traces of the process instances to ensure compliancy to its requirement. The SoD algorithm is composed for this purpose. Where non-compliant behaviour is detected the algorithm returns a violation. The following verification requirements are addressed:

Requirement 4.1: Identify and detect resource assignment violations that lead to role conflicts based on SoD.

Requirement 4.2: Identify and detect roles and tasks upon which SoD violations are likely to occur.

Algorithm 5 SoD Constraint Verification

```
1: $Input$:
       a. All All $Pi$
       b. Constraints (SoD)
2: for all actors (r) where C= SoD do
3:     $(r_1, r_2)$.SoD $\rightarrow (a_1) = (a_1, r_1), (a_1', r_2) \in e.ac$
4:     if $(e.ac) \in seen, finished \neq \quad ((r_1, r_2).SoD)$ then
5:         Violation: SoD constraint violated for $r_1$ and $r_2$ over $(e.ac)$
6:     end if
7: end for
8: Return No violation of SoD constraint for the provided processes.
```

While running, **algorithm 5** checks for all users constrained by the SoD constraint SoD (user) and are assigned to a set of activities. The execution of activities (e.ac) by the constrained resource actors must observe the SoD constraint requirements. The activity events of (c.ac) should exhibit the behaviour to satisfy the constraint. On contrary, if the activity events in the process instances are not the same as the activities described in the behaviour, then the SoD constraint is violated. The behaviour is not seen (SoD user is missing). Otherwise no violation if the same user executed activity event e.ac.

Algorithm for BoD Constraint Verification

Verifying for BoD constraint involves checking the traces in the process instances to ensure compliance with its requirements by the business process. A BoD checking algorithm is composed to detect non- compliant behaviour. The following verification requirements are addressed by the algorithm:

Requirement 5.1: Identify and detect resource assignment violations that may lead to role conflicts based on BoD.

Requirement 5.2: Identify and detect roles and tasks upon which BoD violations are likely to occur to prevent deadlocks.

Algorithm 6 BoD Compliance Verification

```
1: $Input$:
       a. All $Pi$
       b. Constraints (BoD)
2: for all actors (r) with C= BoD do
3:     $(r_1)$.BoD $\rightarrow (a_1, a_2) = (a_1, r_1), (a_2, r_1) \in Pi$
4:     if $(e.ac) \in seen, finished \neq \quad (r_1)$.BoD then
5:         Violation: "BoD constraint violated for $r_1$ over $(e.ac)$"
6:     end if
7: end for
8: Return No violation of BoD constraint for the provided processes.
```

Similar to SoD, if the constraint assigned as part of the activity, the events of that activity should exhibit the behaviour to satisfy the constraint. If the behaviour is not seen (constrained user is missing) then the constraint is violated. Otherwise no violation if the same user executes the assigned activities.

Algorithm for Delegation Constraint Verification

For a role to delegate to another it must have exclusive rights to the activity. Verifying for delegation constraint involves checking the traces in the process instances to ensure that all delegated actors have assumed their responsibilities to prevent task and resource redundancy where resources or tasks become idle, or deadlocks resulting from no resources assigned to execute tasks. A delegation checking algorithm is composed to check non-compliant behaviour. The following verification requirements are addressed by the algorithm:

Requirement 6.1: Verifying that all delegated roles assume their execution responsibilities.

Requirement 6.2: checking for violations likely to lead to role conflicts or idle roles as well as permission leakages.

Delegated users become valid users to execute activities not initially assigned. If a delegated user is not part of the valid user set, or if such users are not the ones that executed the running activities or finished activity set, then the delegation constraint is violated.

Algorithm 7 Delegate Compliance Verification

```
1: $Input$:
       a. All $Pi$
       b. Constraints (Delegate)
2: for all actors (r) where $C = Delegate$ do
3:     $(a_1, r_1)$.Delegate $(r_2) = (a_1, r_1 \wedge r_2) \in e.ac \rightarrow r_2 \in$ Valid Users
4:     if $(e.ac) \in seen, finished \neq$ $((r_1, r_2)$.Delegate) then
5:         Violation: Delegate constraint violated for $r_1$ and $r_2$ over $(a)$
6:     else if $r_2 \notin$ Valid Users then
7:         Violation: "Delegated role not existing"
8:     end if
9: end for
10: Return No violation of Delegate constraint for the provided Business processes
```

5.4. Data Compliance Verification

Verification of compliance with data constraints checks for how a model conforms with data requirements. Such requirements include: data availability and accessibility, Authentication and Privacy. Other requirements forming data constraints include; visibility, interaction and validity security requirements (Russell *et al.*, 2004, 2005; El Gammal *et al.*, 2016). For convenient checking and verification enforcement, the different patterns are compounded into the subcategories discussed below:

1. Data availability and accessibility (AA) constraints: Besides exclusive access requirements, data should be available and accessible to a basic level to facilitate work progress. Besides, data should be available and accessible whenever required. Verification of AA constraint requires checking for compliance with availability and accessibility data requirements.
2. Data Privacy constraint: the requirement to observe privacy of data justifies the establishment of access control and authorization. Privacy constraint originates from the GDPR data privacy

principle where organisations are required to build data privacy as part of their systems. Verifying for data privacy involves checking for enforcement of privacy controls over data.

3. Authentication constraint: Authentication is a constraint to achieve basic security of data and systems by requiring users to be identified and given access. Authentication involves the process of validating the identity of a registered user before allowing access to the protected resource. As a data constraint, authentication restricts access to data by requiring prior user login and profile authentication. It is based on identity management where digital identities are managed based on organisational security policies to ensure that only necessary and relevant data is shared using user identity and profile data as well as data governance functions.

Similar to privacy, compliancy to security constraint is demanded by many regulatory standards like GDPR and Anti money laundering. Specifically, GDPR emphasizes security by design. Integrating security constraints and checking for their compliance in the process model is therefore important to meet policy and regulatory requirements.

5.4.1. Specification of Data Constraints

Boolean conditions are used to evaluate data access conditions are true or false. Depending on the outcome, access is granted or denied. If a trace is true to the conditions specified, then it satisfies the constraint. Otherwise it is false and violates the constraint. To that effect, the following specifications and definitions are useful for the data checking algorithm. Given a set of activities a_1, a_2 and a_3, assigned to resource actor (r_1) and requires access to product catalogue data (Pcd). Access to this data is constrained by access and availability, i.e. only 'Read' action can be granted. If the assignment is true according to the executed behaviour, then the trace (σ) satisfies (|=) the constraint.

Definition 5.4.1. Accessibility and Availability (AA)

$\sigma \in \left(\left((a_1, a_2, a_3), r_1\right):(Pcd.[Read]):AA\right)$

If $(\sigma = True)$ then $\sigma \vDash AA$

The definition specifies accessibility and availability constraints for Pcd data object with action read granted to r_1 for execution of activities a_1, a_2, and a_3. During verification, the data compliance verification algorithm checks for compliance to the constraint for the data object, action by the user and tasks. If the outcome shows that the trace is true to the constraint requirement, then the trace satisfies the availability and accessibility constraint. Otherwise, it is a violation detected for the AA constraint.

Definition 5.4.2. Authentication

$\sigma \in \left(\left((a_1, a_2, a_3), r_1\right):(Pcd.[True|False]):Authentication\right)$

If $(\sigma = True)$ then $\sigma \vDash Authentication$

The definition specifies access control by authentication granted for accessing Pcd data with actions to read and write for role actor (r1) who executes activities a1, a2 and a3. Satisfaction of the authentication constraint is achieved if the traces of the executed events show exhibit the specified behaviour. Otherwise, a violation is detected for the authentication constraint.

Definition 5.4.3. Privacy (Prv)

$\sigma \in \left(\left((a1, a2, a3), r1\right), Pcd.[Read]\right):Prv)$

If $(\sigma = True)$ then $\sigma \vDash Prv$

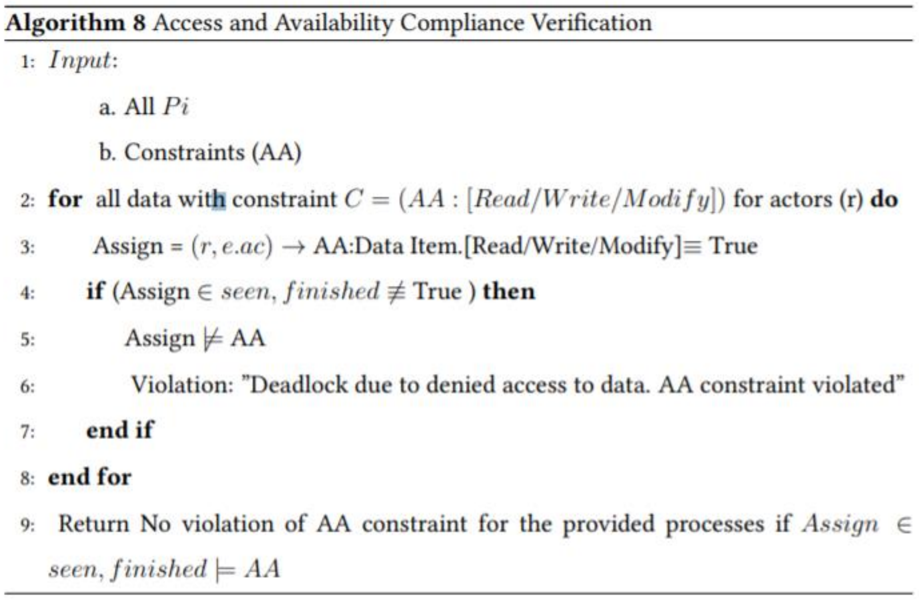
The definition specifies Privacy constraint for accessing Pcd data where action to read private data is to be granted to the resource actor r1 who executes activities a1, a2 and a3. During verification, the privacy compliance verification algorithm checks the constraint for its satisfaction before access can be granted to read private data. If the trace is true for the specification, then the constraint is satisfied and thus compliance achieved. Otherwise, it is a violation detected for the privacy constraint.

Algorithm for Access and Availability Constraint Verification

Verifying for data access and availability Constraints ensures that basic non-exclusive data is accessible and available with less restriction to enable accomplishment of basic tasks. Algorithm 8 is composed to the effect. Violation occurs if role actors or tasks are denied access to data constrained by AA or where the permitted action type differs from the initial assignment, e.g. modify action type instead of read action type. The verification requirements addressed by algorithm 8 are:

Requirement 7.1: Ensure that required data is available and accessible for all tasks and role actors as required by AA constraint. This prevents events from executing without access to data. This prevents deadlocks where running events have no access to data or data is not available and events keep waiting for it.

Requirement 7.2: Identify and detect AA constraint violations likely to lead into data access denial.



Algorithm 8 Access and Availability Compliance Verification

```
1: $Input$:
       a. All $Pi$
       b. Constraints (AA)
2: for all data with constraint $C = (AA : [Read/Write/Modify])$ for actors (r) do
3:     Assign = $(r, e.ac) \rightarrow$ AA:Data Item.[Read/Write/Modify]$\equiv$ True
4:     if (Assign $\in$ $seen, finished \not\equiv$ True ) then
5:         Assign $\not\models$ AA
6:         Violation: "Deadlock due to denied access to data. AA constraint violated"
7:     end if
8: end for
9: Return No violation of AA constraint for the provided processes if $Assign \in$
   $seen, finished \models AA$
```

Violation of AA constraint as per **algorithm 8** exists when tasks or their actors (r, e.ac) are denied access to data whose constraint is AA. This violation leads to a deadlock or livelock. Deadlock occurs if running activities are denied access to data necessary for the process to continue in execution. Whereas the livelock occurs when a task is denied access to data stays in waiting mode stagnating process execution. The other form of violation may occur when the activity finishes execution without necessary data. This leads to wrong outcomes which do not comply with specifications.

Algorithm for Verifying Compliancy with Authentication Constraint

Authentication verification **algorithm 9** verifies for compliance by checking that role actor credentials match the credentials stored in a database of authorized actors as well as the database for access privileges over tasks. The algorithm checks for three forms of Authentication errors which are the sources of authentication related violations:

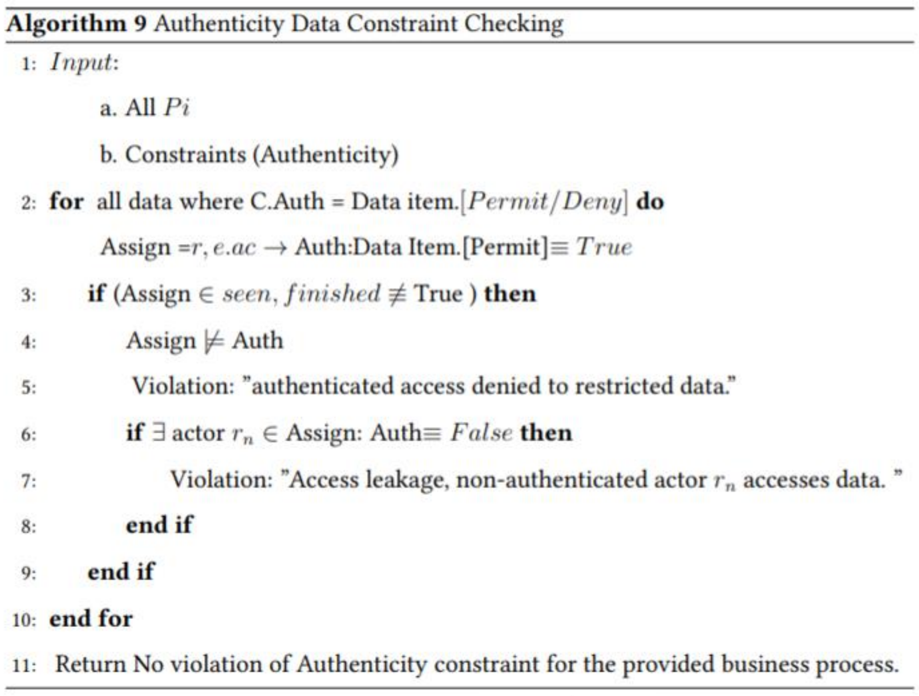
- Access leakage which occurs when non-authenticated users gain access to data.
- Deadlocks which occur when users are authorized to execute activities but access to data is denied for technical or logical reasons e.g. improper configurations. •

- Authentication breach which occurs when non-authenticated activities or users intentionally gain access to data. This is traced from running or finished events.

The following verification requirements are addressed by the algorithm:

Requirement 8.1: Prevent security lapses or leakages by checking actor identify and detect unauthenticated access to data by task executors or roles.

Requirement 8.2: Detect authentication violations upon tasks based on access types.



Algorithm 9 Authenticity Data Constraint Checking

```
1: $Input$:
      a. All $Pi$
      b. Constraints (Authenticity)
2: for all data where C.Auth = Data item.[$Permit/Deny$] do
      Assign =$r, e.ac \rightarrow$ Auth:Data Item.[Permit]$\equiv True$
3:    if (Assign $\in seen, finished \not\equiv$ True ) then
4:       Assign $\not\models$ Auth
5:        Violation: "authenticated access denied to restricted data."
6:       if $\exists$ actor $r_n \in$ Assign: Auth$\equiv False$ then
7:          Violation: "Access leakage, non-authenticated actor $r_n$ accesses data. "
8:       end if
9:    end if
10: end for
11: Return No violation of Authenticity constraint for the provided business process.
```

($Assign \not\models Auth$).

Actors are permitted or denied access to data by authentication. Where data constrained by authenticity is accessed by non-authenticated actors, it implies access leakage i.e. data is accessed by actors without authentication. Similarly, where access to data is denied to authentic actors, it leads to a deadlock since they cannot execute the current work in progress. Authenticity compliancy checking algorithm checks for permitted or denied access to restricted data based on actor identities and roles. Where the assignment to data does not match the prescribed access policies, a violation is detected. Similarly, violations are identified from traces where transactions have occurred if the assignment does not match the traces

Algorithm for Verifying Compliancy with Privacy Constraint

Privacy constraint is enforced by means of access control and authorization. Authorization involves the process of validating that the authenticated user is granted permission to access the requested resources. Privacy as a data constraint restricts access to data regarded private as defined by GDPR. Data that is not available to the public is accessible by fulfilling authorization requirement. Violation to privacy constraint is checked targeting two forms of errors; deadlocks and privacy breach.

- Deadlocks occur when the executing events authorized to access data are denied access for technical or logical reasons e.g. improper configurations,
- Breach to privacy i.e. non-authorized activities eventually access private data and execute.

To verify for these errors in a business process, algorithm 10 is composed. Authorized actors are granted permission to Read/Write/Modify private data items. Therefore, compliant traces or transactions are those where the Assignment is equivalent to the authorized actions (Assign ≡ Authorize). Violations are detected or identified in traces where authorized permissions differ from the assigned (Assign ≠ Authorize).

Algorithm 10 Privacy Data Constraint Checking

```
1: $Input$:
        a. All $Pi$
        b. Constraints (Privacy)
2: for all data where C=Privacy:[Read/Write/Modify] for actors $(r)$ do
3:     Assign= $(r, e.ac) \rightarrow$ Privacy: Data Item.[Read/Write/Modify] $\equiv$ Authorise
4:     if $(Assign \in seen, finished) \not\equiv$ Authorise then
5:         Assign $\not\models$ Privacy
6:         Violation: "Authorised actors denied access to private data"
7:         if $r_n \notin (r, e.ac) | r_n \in$ Authorise then
8:             Violation: "Access leakage, non authorised actor access to private data"
9:         end if
10:    end if
11: end for
12: Return no violation of Privacy constraint for the processes if $Assign \models Privacy$
```

The other form of violation is where privacy constrained data exists outside the restricted boundary. This leads to a leakage since it is accessible by non-authorized actors. Similarly, where authorized data is not visible in ‘seen’ and ’finished’ events it signifies a violation in form of a deadlock where data was not available or accessible to facilitate task execution. Authentication and privacy constraints are enforced by means of process driven access control and authorization (PDAC) (Kasse *et al.*, 2020). **Section 6** discusses the PDAC concept in detail.

Overall Compliance Verification Algorithm

The overall compliance verification algorithm is a general algorithm that integrates the specific constraint checking algorithms into a single algorithm to check the entire business process behaviour. The application of this algorithm is twofold:

- It can be applied to verify a business process where a large amount of modifications has been made necessitating checking the entire model for constraints compliancy, or
- Where a business process is designed from scratch automatically requiring full scale verification for compliance with policy and regulatory requirements.

Algorithm 11 Overall Compliance Constraint Verification Algorithm

```
1: $Input$:
       a. All $Pi$
       b. All Constraints
2: for all $Pi$:C = Control flow, Resource, Data, and Temporal do
3:     Verify compliance with control flow constraints
       Trace validity → Call algorithm 1
       Existence → Call algorithm 2
       Precedence → Call algorithm 3
       Response → Call algorithm 4
4:     if $Pi \models C = True$ then
5:         Verify compliance with Resource constraints
       Check SoD → call algorithm 5
       Check BoD → call algorithm 6
       Check Delegate → call algorithm 7
6:         Verify compliance with Data constraints
       Check AA → call algorithm 8
       Check Auth → call algorithm 9
       Check Privacy → call algorithm 10
7:         Message = Compliance status for Control flow, Resource, Data constraints
8:         Return overall compliance feedback for the provided business process.
```

6. Process Driven Access Control and Authorisation (PDAC)

PDAC is a concept proposed in (Kasse *et al.*, 2018, 2020) as a mechanism towards realization of an automated and agile, yet less complex solution to overcome the challenges of non-compliance to security and privacy constraints. The motivation and rationale were based on the compliancy demands of the 2018 revised GDPR. At the dawn of the May 2018 launch of the revised GDPR version, big companies like Facebook, Inc. (Patterson, 2020) and Google LLC (Satariano, 2019) were already faulted for data privacy breaches. The GDPR articles of interest to this study are the principles of security by design and privacy by design. 'The former principle requires security of the data to be built within the information system design. The latter principle requires transparency from the data protector and processor to make known to the data owner the status of their data i.e. when it is being collected, processed and transmitted. Before collection and processing, the data owner's consent must be sought.

PDAC leverages existing solutions to enhance access control and authorizations to achieve automated compliancy, especially with dynamic policies and regulations. It ensures regulated and legalized data access based on its need to accomplish a specific process instance. As a divergent access control mechanism from existing access control mechanisms, access under PDAC is based on the entire process instance by assessing the purpose, time and instance as opposed to the subject, object or action to be committed. This is a paradigm shift from the traditional access control models based on tasks (Thomas and Sandhu, 1994), roles (Thomas and Sandhu, 1994; Sandhu, 1995; Ferraiolo *et al.*, 2001) and attributes (Jin, Krishnan and Sandhu, 2012; Hu *et al.*, 2014, 2015) which grant and authorize more access than what is required. This violates the data privacy principle.

Despite their role in security and privacy administration, classical access control mechanisms are unable to support modelling and enforcement of security and privacy requirements presented by current workflows which must as well comply with many other regulations. Relatedly, workflows supporting collaborative business processes present more complex and dynamic security and privacy requirements that require agility to implement which is not provided in the current mechanisms. They grant roles more authority and permissions beyond what may be required.

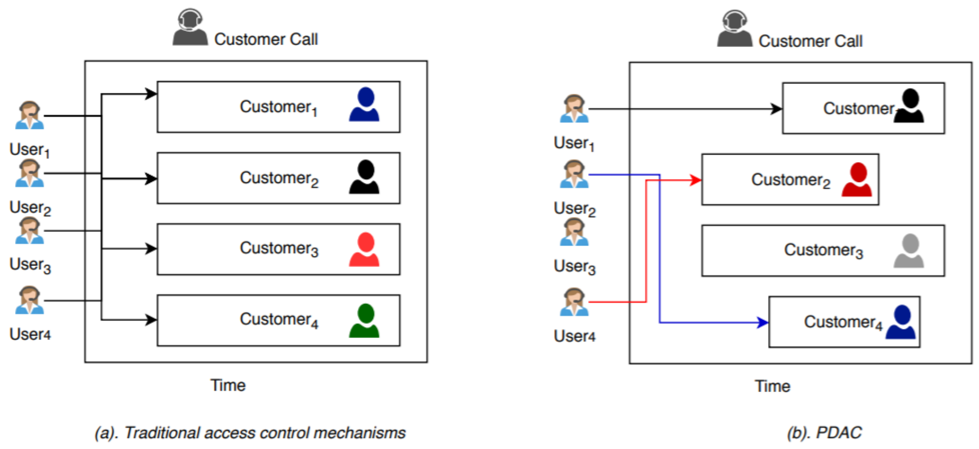


Figure 8. Illustration of PDAC vs. Traditional access control mechanisms

Figure 8 part (a) illustrates authorized users in a call centre granted full access to all customer records indiscriminately. They have access to records all time. Part (b) illustrates PDAC where users are granted access to a single record per session of time a customer is being served. Various extensions to the classical access control mechanisms have been suggested. In **Table 9,** a summarized description of mechanism extension is presented together with PDAC. It is noticeable that the most common constraints dealt with are SOD and BoD. The suggested PDAC mechanism differs from the classical ones to address privacy and authentication constraints

Proposal	**Constraints**	**Mechanism**	**Output**	**State**
Support dynamic assignment of access controls based on the task instance context and task states	DSOD, BOD, Temporal constraints	BAC and RBAC	AC agent enforcement architecture	Design time, Runtime
Support modelling of constrained workflows for local and global constraints such that a sound workflow constrained schema exists where authorised user can execute a complete workflow instance	SOD, BOD, cardinality constraints	TBAC and RBAC	Formalised constrained sound workflow	Design time
The management of authorisations of organisation roles in a process view	SoD, duty of conflict	TBAC and RBAC	Algorithm	Design time
Authorisation and Access control model for giving subject access to objects during task execution	No concern for SoD or BoD	RBAC	Authorisation and access control Model	Runtime
A privacy-aware BP modelling framework supporting reasoning and enforcement of privacy constraints	Separation of tasks, Binding of Tasks, Necessity to know	User Roles	Extension of BPMN 2.0 to PrVBPMN	Design time
PDAC - Support process driven access control and authorisation	**Privacy, authentication and security constraints**	**Process Instance, Time**	**Compliance verification Algorithm**	**Hybrid**

Table 9. Research on extensions of Access control mechanisms

6.1. Implementation architecture for Process Driven Access Control and Authorization

Access to data is granted by authorization and revoked automatically in two ways i.e. i) Once the purpose for which access was granted is accomplished, and ii) When the assigned duration expires. In either case, the resource actor ceases to have access to data. For example, in **Figure 9** user is assigned access to a single customer's data for an instance of a call and access will cease the moment the call ends. During execution, when access to data is required, the authorization service is invoked to check the assigned access privileges. It then provides feedback for granted or denied access and provide message to the user via the dashboard.

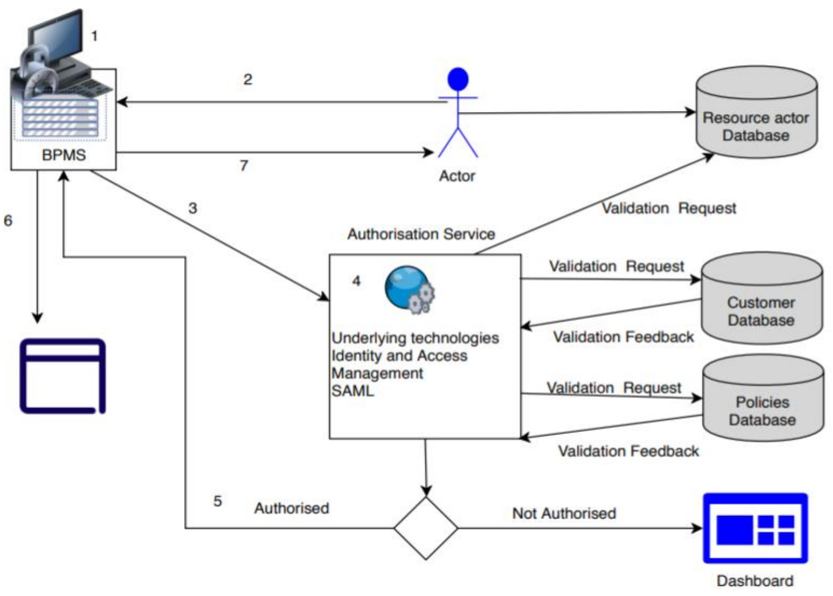


Figure 9. PDAC Authorization Service Architecture

1. Activity started
2. User accepts tasks
3. BPMS work list handlers' issues data authorization token
4. Authorisation engine validates request token with policy and customer databases
5. Token validated and issued to BPMS
6. The token is stored in the browser/ user client
7. Actor executes activity

Within the business process management system an activity event is initiated as step (1) shows the activity is then assigned to a resource actor which will accept it in step two (2). The activity now exists in the work list of the actor (system user) in the BPMS. The BPMS issues an authorization token request to access the required data in step (3). In step (4) the authorization service is managed by the authorization engine implemented by underlying technologies like identity and access management (IAM) and Security Assertion Markup Language (SAML).The authorization involves validation of the request against user identities, policies and customer data in their specific databases. A collection and validation of a combination of these parameters legitimizes access authorization. The token is validated either offline with a short duration session token or with digital signature online validation. In step (5) a validated token is returned to the BPMS authorizing activity execution by the actor and stored in the browser or user client profile in steps (6) and (7).

6.2. User Authentication

SAML technology supports enforcement of user identification and authentication. The user signs into the client portal e.g. a browser which sends an authentication request to the user identity database. The database authenticates the user by generating SAML authentication assertions that identify the users and their information. The browser contacts the validation service with the SAML assertion which requests temporary security credentials and creates session for sign in. The sign in is sent to the browser granting access to the users based on policies in the policy database.

6.3. GDPR Implementation

The customer self-service point is for implementation and fulfilment of GDPR requirements. Enforcing compliance to GDPR requirements is achieved by enabling:

- Data owners have access to personal data by means of automated access.
- Restrict processing of data by data owners by directly interacting with data processors.
- Data modification and deletion through a self-service interface.
- Data portability to enable data transfer serviced by the data owner.
- Audit and monitoring of data by its owner at any point in time.

7. Compliance Checking and Verification with Use Case

This section presents the application of the artifacts, i.e. the compliance verification algorithms to check the compliance of a business process with the required constraints. The formalization and the design of the compliance verification algorithm followed a stepwise approach based on use case 1 which was described in section 3. To demonstrate artifact applicability, we still apply the use case 1 but in a different way. For this purpose, understandability and space reasons, use case 1 is abstracted to represent internal process operations of the store, and verified using the overall compliance verification algorithm in section 8.4 specifically, the order processing instance is considered.

7.1. The Abstracted Pick and Pack Use Case

The process starts with arrival of orders in the store's order catalogue. The orders are sorted, assigned and processed to completion. The order processing Eco system is composed of the orders, customers, staff, policies and regulations, and regulatory agencies among others. These play different roles:

- Orders are placed by customers and pick them when they are ready or wait for delivery.
- Staff process orders at the store e.g. Pickers, Packers, supervisors among others.
- Policies and Regulations guide operations of the business process.
- Regulatory agencies specify and monitor enforcement of policies and regulations.

The activities in the abstracted pick and pack business process are briefly described as follows:

- Select Order (So): the order is selected from the pending orders by a staff who will process it. This is the initial activity which signals the start of order processing instance.
- Pick items (Pit): The items are picked by the store staff. A store may have one or more store departments and staff may cross between departments or are restricted to one.
- Verify order (Vo): This is a quality check to ensure the order is fulfilled in terms of the right items and quantities.
- Pack order (Po): The order is packed and made ready for delivery or pickup by the customer.
- Hand over (Ho): The ready order is handed over to customer service unit
- Customer Pick up or Delivery (Cpd): if the order is not picked up the delivery, staff will deliver the item within the specified duration.

Based on the process activity brief description above, consequently the model in **Figure 10** is realized.

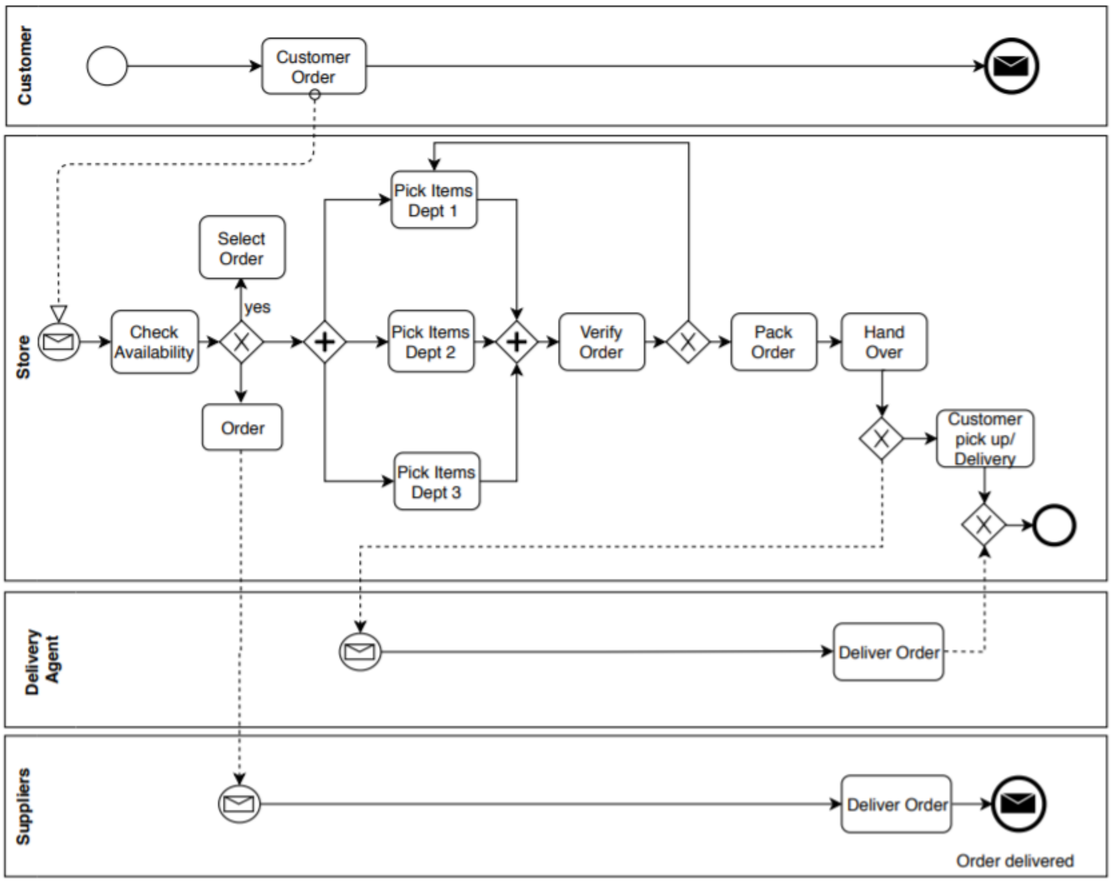


Figure 10: Abstracted pick and pack business process model

7.1.1. The Internal Requirements of the Business Process

As described, the business process must conform to a set of policies specific for a store. Some of the relevant policies include:

Control flow and temporal policies to guide process executions are as follows:

1. Each order must start with the select order activity and end with customer pick up or delivery. The total order processing time is 3 hours.
2. During order processing, big orders are picked by more than one staff. This activity duration should not exceed one hour.
3. Every order must be verified before it is packed. Verification of each order depending on the size within 20 minutes.
4. Packed orders are ready for handover to customer service section
5. Orders are picked by customers or delivered to customer premises. Delivery takes one hour whereas the customers must pick their orders within a day otherwise they are put in the storage.

In addition, resource-based policies to guide allocation resources are as follows:

- Pickers are allocated to pick items and cannot execute verify orders.
- Packers are allocated to pack order task. However, they also execute verify order task.
- Pickers can be delegated to participate in order hand over to customers if they are free or when there are high volumes.
- Supervisors oversee other employees and can execute any task.
- Supervisors can execute delegate tasks. E.g. supervisors can delegate pickers to pack items.

The specified tasks are executed if access to necessary data is provided. To this effect, policies to guide access control to data are specified as follows:

- Supervisors have full access to data and can grant data access to staff based on organisational roles and tasks they execute in the business process.
- Basic data must be accessible and available for staff to execute tasks that do not need much restriction and control. For example, order list data should be accessible and available to pickers, verifiers and packers.
- Access control and authorization must be observed for data privacy. For example, customer personal data, financial data among others
- Customer data is considered as private data to which the principle of privacy must be observed.
- Security of the data and system is important and worth observation. To this effect, users and staff must be authenticated to use the system.

The internal policies are superseded by the external regulations. The super store being cross regional, several external regulations apply. Such as:

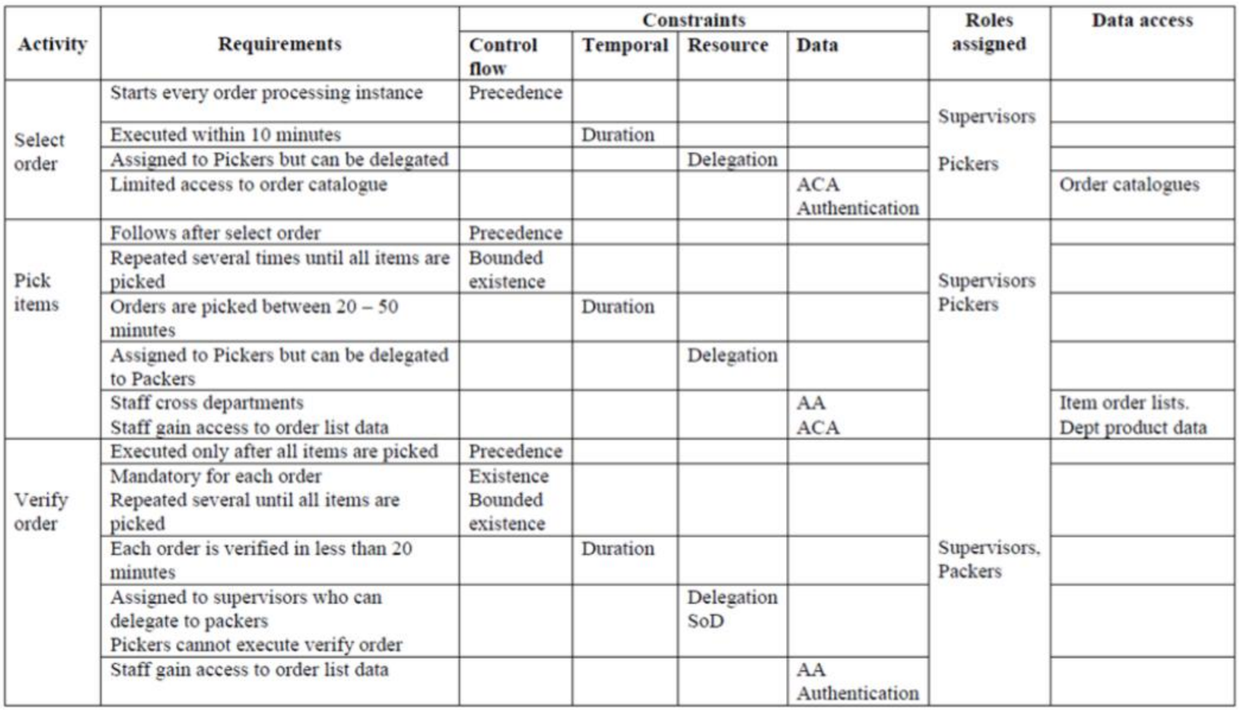
- The European union general data protection act (GDPR) which emphasizes data privacy and security
- The Sarbanes Oxley Act (SOX) which emphasizes the separation of duty and binding of duty.
- The UK consumer protection act which emphasizes consumer protection rights like right to quality products and services, right to return goods, right to be refunded.
- The Health Insurance Portability and Accountability Act (HIPAA) or the NHS equivalent which defines basic security and privacy practices for health care and pharmaceutical dispensaries. The act applies to the stores since many of them operate pharmacies.
- Trade laws limiting sale of restricted products to specific groups of customers like those in underage category. For example, sale of alcoholic products. Also, sale of health products that require drug prescriptions.
- Service level agreements for acceptable business transactions and customer relations.

Both internal policies and external regulations must be complied with by the business process. Because of the collaboration, contractual obligations are composed and agreed upon by the parties as guiding principles for business operations. A collection of requirements from applicable policies, rules, laws, standards and regulations forms a set of all compliance requirements that the business process must conform with. This document is updated as change in policies and regulations occur.

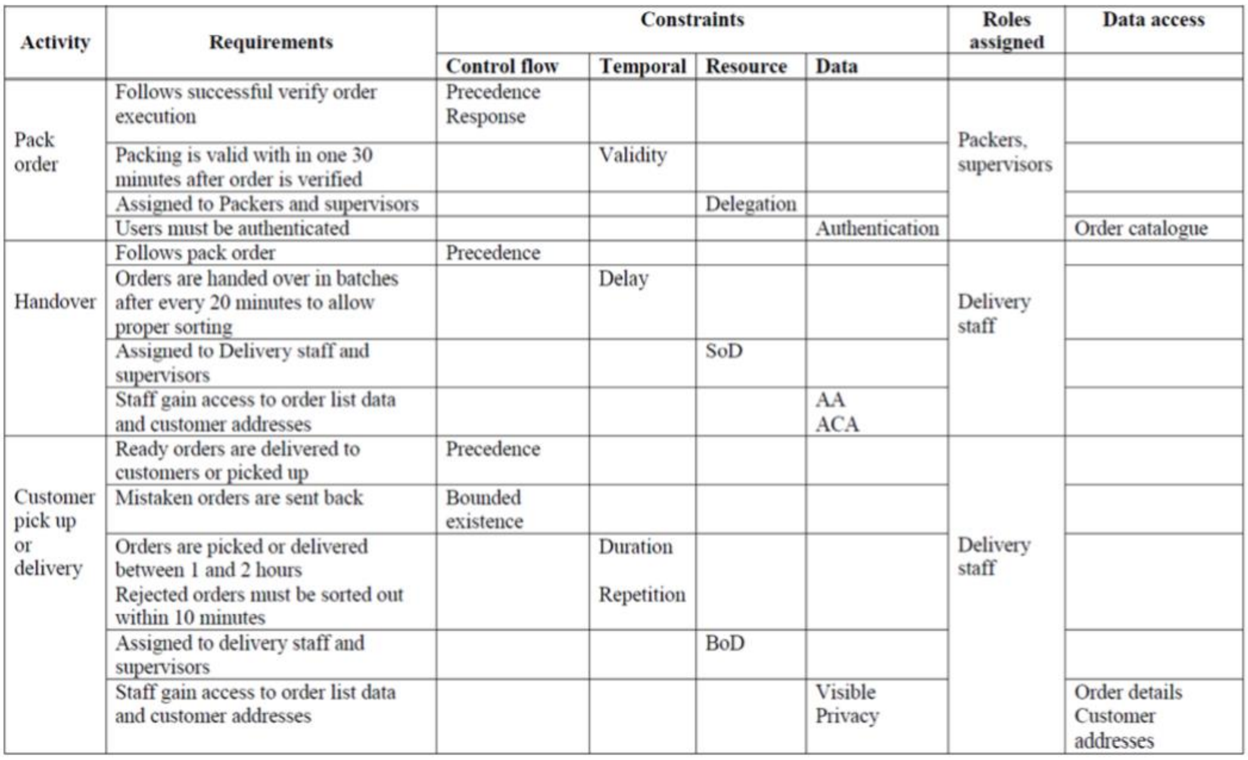
As earlier indicated, policies and regulations are stated in natural language and thus bound to suffer the challenges of natural language such as ambiguities and inconsistency. The extracted requirements form the compliance constraints that are verified with the business process model. The verification is only possible with formalized constraints. From this point, the artifacts put forward by this paper are applied. In the next sections, the application of constraint expression mechanism is illustrated.

7.1.1. Constraint Elicitation and Expressions

In consideration of the above, a list of requirements and constraints are for the pick and pack process as presented in **Table 10** below. The next step is to formalize the listed constraints through formal specifications in section 7.2 based on DL.



Activity	Requirements	Constraints				Roles assigned	Data access
		Control flow	Temporal	Resource	Data		
Select order	Starts every order processing instance	Precedence				Supervisors Pickers	
	Executed within 10 minutes		Duration				
	Assigned to Pickers but can be delegated			Delegation			
	Limited access to order catalogue				ACA Authentication		Order catalogues
Pick items	Follows after select order	Precedence				Supervisors Pickers	
	Repeated several times until all items are picked	Bounded existence					
	Orders are picked between 20 – 50 minutes		Duration				
	Assigned to Pickers but can be delegated to Packers			Delegation			
	Staff cross departments Staff gain access to order list data				AA ACA		Item order lists. Dept product data
Verify order	Executed only after all items are picked	Precedence				Supervisors, Packers	
	Mandatory for each order Repeated several until all items are picked	Existence Bounded existence					
	Each order is verified in less than 20 minutes		Duration				
	Assigned to supervisors who can delegate to packers Pickers cannot execute verify order			Delegation SoD			
	Staff gain access to order list data				AA Authentication		



Activity	Requirements	Constraints				Roles assigned	Data access
		Control flow	Temporal	Resource	Data		
Pack order	Follows successful verify order execution	Precedence Response				Packers, supervisors	
	Packing is valid with in one 30 minutes after order is verified		Validity				
	Assigned to Packers and supervisors			Delegation			
	Users must be authenticated				Authentication		Order catalogue
Handover	Follows pack order	Precedence				Delivery staff	
	Orders are handed over in batches after every 20 minutes to allow proper sorting		Delay				
	Assigned to Delivery staff and supervisors			SoD			
	Staff gain access to order list data and customer addresses				AA ACA		
Customer pick up or delivery	Ready orders are delivered to customers or picked up	Precedence				Delivery staff	
	Mistaken orders are sent back	Bounded existence					
	Orders are picked or delivered between 1 and 2 hours Rejected orders must be sorted out within 10 minutes		Duration Repetition				
	Assigned to delivery staff and supervisors			BoD			
	Staff gain access to order list data and customer addresses				Visible Privacy		Order details Customer addresses

Table 10. Requirements and Constraint Lists

Requirement Expressions

DL Based Specification

This section illustrates requirements representations using DL based on the constraint expression mechanism describe in section 4. The symbols used include:

- u Conjunction of constraints
- t Disjunction of constraints

- → Assignment of an activity to a constraint
- : Assignment of subsequent constraints after the initial (control flow) constraint
- [,] Brackets holding constraint attributes

Constraint Representations using Unary Expressions

The unary expressions represent individual category-based constraints:

1. Example control flow and temporal constraint expressions Requirement 1 specifying that the select order activity Starts every order processing instance, executed within 10 minutes, assigned to Pickers but can be delegated and data access is limited access to order catalogue. 'is requirement can be expressed as follows:

   So → (Exist) ∩ Duration: (10mins)

   Pit → [So] Precede ∩ BoundedExit ($^{n-1}$) ∩ Duration: (20 − 50mins)

   Vo → [Pit] Precede ∩ BoundedExit[n] → Duration: (≤ 20mins)

   P o → [Vo] Response ∩ Precede ∩ Valid: (10mins)

   Ho → [Po] Precede ∩ Delay :(20mins)

   Cpd → [Ho] Precede ∩ BoundedExit[n] ∩ (Duration: [1−2hrs] ∩ Repetition: [10mins])

2. Example Resource constraint expressions

   So → (Supervisor) ∩ Delegate: (Supervisor → Pickers)

   Pit → (Pickers, Supervisors) ∩ Delegate: (Supervisor → Packers)

   Vo → SoD: (Supervisors, ¬Pickers) ∩ Delegate: (Supervisor)

   Po → BoD: (Supervisors, Packers)

   Ho → BoD (Supervisors, Deliverystaff)

   Cpd → BoD (Supervisors, Deliverystaff)

3. Example Data constraint expressions

   So → ACA ∩ Authentication: (Ordercatalogue)

   Pit → AA: (Itemorderlists) ∩ ACA: (Departmentitemlists)

   Vo → ACAAuthentication: (Itemorderlists)

   P o → Authentication (Ordercatalogue)

   Ho: (Ordercatalogue)

   Cpd → Visible ∩ AA: (Ordercatalogue)Privacy: (Customeraddress)

Constraint Representations Using Binary Expressions Binary expressions are composite representations involving combinations between sets of constraints. The requirements in Table ‰ involve combinations of constraints that guide execution behaviour. This subsection illustrates expression of requirements involving binary constraints per activity.

1. Select order execution constraints expression

   $$So \rightarrow (Exist \cap \neg Precede) \cap Duration{:}[<10mins] BoD[Picker] \cap Itemorderlist[Auth] \cap [ACA]$$

   Requirement 1 specifying that the select order activity starts every order processing instance, executed within 10 minutes, assigned to Pickers as BoD but can be delegated and data access is limited access to order catalogue by access control and authorization.

2. Expressions of Pick items execution requirements

   $Pit \rightarrow (\neg Exist[So] \sqcap BoundedExist[n_{n-1}]) \sqcap Duration: [20 - 50 Mins]$
   $\sqcap (BoD: [Picker] \sqcap \left[Delegate: (Supervisors, Picker, Packer)\right)$
   $\sqcap\ itemorderlist: [AA] \sqcap [ACA]$

   The expression specifies that pick items activity is preceded by select order and can be repeated several times until all items on the order list are picked. The scheduled duration is between 20 and 50 minutes, with a BoD resource constraint for the picker, and access to item order list data granted by access and availability, and by access control and authorization.

3. Expressions of Verify order execution requirements

   $Vo \rightarrow (Precede[Pit] \sqcap BoundedExist[n_{n-1}]) \sqcap Duration: [< 20 Mins]$
   $\sqcap \left(SoD: [\neg Pickers] \sqcap Delegate: (Supervisors, Packer)\right)$
   $\sqcap\ itemorderlist: ([AA] \sqcap [Auth])$

   The expression specifies that verify order activity is preceded by Pick items and its conditions must be satisfied before the process continues to the next level which implies that it is repeated several times. The scheduled duration is less than 20 minutes, with SoD resource constraint for the pickers and supervisor who can delegate to pickers. Access to item order list data is granted by authentication, and by access control and authorization.

4. Pack Order execution constraints expression

   $Po \rightarrow (Precede[Vo] \sqcap Response) \sqcap Valid[=30 Mins] \sqcap (BoD: [Packers] \sqcap$
   $Delegate[Supervisors, Picker] \sqcap itemorderlist: ([AA] \sqcap [Auth])$

   The expression specifies that pack order activity is preceded by verify order and occurs as a response to verify order. Its execution is valid for 30 minutes. The assigned resource constraint is BoD for the packers and supervisor who can delegate to pickers. Access to item order list data is granted by accessibility and availability, and access control and authorization.

5. Handover Order execution constraints expression

   $Ho \rightarrow (Exist \cap Precede[Po]) \cap Delay[20 Mins] \cap Role:[Supervisors, DeliveryStaff]$
   $\cap Itemorderlists[AA] \cap [ACA]$

   The expression specifies that handover order activity is preceded by Pack order. Its execution is delayed for 30 minutes to allow batch processing of handover. The assigned resources are supervisors and delivery staff. Access to item order list data is granted by accessibility and availability, and by authentication.

6. Customer pick-up or Delivery execution constraints expression

   Cpd → (Exist ∩ Precede[Po]) ∩ (Duration:[1−2HoursMins] ∩ Repetition[10mins]) ∩ [Supervisors, DeliveryStaff] ∩ (Itemorderlists : [AA], customeraddresses : ∩ [ACA]) The expression specifies that order delivery or customer pick-up activity is preceded by handover order, executed for a duration of 1-2 hours and it is repeated every 10 minutes in case the order is rejected. The assigned resources are supervisors and delivery staff with access to order list data granted by accessibility and availability, while customer address data is granted by satisfying privacy data constraints.

Example Formal Constraints

To enhance the reasoning capacity, DL was extended with integration of basic constructs of LTL i.e. operators and quantifiers to obtain more formal constraint expressions. The model logic created facilitates compliance verification and checking of business process and constraints. The section below presents the example formal expressions.

1. Select order execution constraint expression

   $G(So[init] \wedge [< 10 mins] \wedge [picker, Supervisor: BoD] \wedge [Itemorderlist: (AA, Auth)]$

The expression specifies *So* as an initial activity whose duration is less than 10 minutes. It is assigned to pickers and supervisor as resources constrained by BoD which implies that the picker can participate in another activity. Access to item order data is controlled by access and availability as well as authentication.

2. Pick Items execution constraint expression

$$G(Pit^{n} \xrightarrow{n-1} \wedge [20 - 50Mins] \wedge [Picker{:}\, BoD, (Supervisors, Packer{:}\, Delegate)]$$
$$\wedge [itemorderlist{:}\, (AA, ACA)]$$

The expression specifies Pit as an activity that can be repeated for n times, for duration between 20-50 minutes. It is assigned to pickers and supervisor as resources constrained by BoD which implies that the picker can participate in another activity. The supervisor can delegate task to packers. Access to item order list data is controlled by access and availability as well as authentication.

3. Verify order execution requirements

$$G(Vo^{n} \xrightarrow{n-1} \wedge [< 20mins] \wedge [Verifiers[SoD])(Supervisors, Packers{:}\, [Delegate])$$
$$\wedge\, itemorderlist{:}\, (AA, Auth)])$$

The expression specifies *Vo* as an activity that can be repeated for n times until it passes, for a duration between of less than 20 minutes. It is assigned to packers as resource constrained by SoD. ‘e supervisor can delegate task to packers. Access to item order list data is controlled by access and availability as well as authentication.

4. Pack Order execution constraint expression

$$G(Po \rightarrow \wedge [30Mins] \wedge [Packers{:}\, BoD(Supervisors, Picker{:}\, Delegate)] \wedge$$
$$[itemorderlist]{:}\, (AA, Auth))$$

The expression specifies Po as an activity to be executed for duration of 30 minutes or less by packers and supervisor as resources constrained by BoD which implies that the packers execute Po in relation to another activity. The supervisor can delegate the activity to pickers. Access to item order list data is controlled by access and availability as well as authentication.

5. Handover Order execution constraint expression

$$G(Ho \rightarrow [20Mins] \wedge [(Supervisors), Pickers{:}\, Delegate] \wedge [itermorderlist{:}\, (AA, ACA)])$$

The expression specifies *Ho* as an activity scheduled for duration of 20 minutes. It is assigned to supervisors who can delegate to pickers. Access to item order list data is controlled by access and availability as well as authentication.

6. Customer pick-up or Delivery execution constraint expression

$$G(Cpd \rightarrow \wedge [1 - 2HoursMins, 10Mins] \wedge [Supervisors, DeliveryStaff]$$
$$\wedge [itemorderlist{:}\, AA, customeraddresses{:}\, ACA])$$

The expression specifies Cpd as an activity scheduled for duration between 1- 2 hours. It is assigned to supervisors and delivery staff. Access to item order list data is controlled by access and availability while customer addresses data is controlled by privacy constraint as well as authentication

7. if Duration >=24 hours then Action “Take package to store”

When the orders are not picked for the day, they are taken to the store for storage. The expressions in this section demonstrate the converted formal expressions making use of binary relations among the constraints to specify behaviour of the process. To illustrate the reasoning, a set of verification requirements are specified as follows:

Verification Scenario – Requirements

In this scenario, the following verification requirements are listed, their specification and formal expressions:

1. Every order processing instance starts with select order and ends with delivery or customer pick up.

   $G((So), F(Cpd))$

   For purpose of checking termination of instances, each terminating case starts with selects order and ends with order delivery or pickup.

2. Every order processing instance must be verified. Verify order must exist in every instance.

   $G(\forall\sigma \in Pi\ \exists Vo)$

   For every case of order processing instance must always be verified

3. Supervisors have rights to every task and can delegate tasks to other users.

   $G(\forall Activities, Supervisor \rightarrow (ACA.[Read]) \wedge F(Delegate))$

   For each activity, always the supervisor has access control and authorization, and can eventually delegate permissions.

4. A set of activities are BoD and SoD respectively

   $G((Pickers, Supervisors).BoD \rightarrow (So, Pit)$

   Activities select order and Pick item are always executed by resource actors pickers and supervisors constrained as BoD. ‘ e roles meet resource actors selection conditions for the execution of So and Pit.

   $G((Verifiers, Supervisors) \wedge (\neg Pickers).SoD \rightarrow (Vo)$

   Activity verifies order is always executed by verifiers or supervisors as designated role actors that meet resource selection conditions for its execution. Pickers are excluded from roles that can execute verify order.

5. Verify Order must wait until Pick order is completed. Pick order is repeated until all items are picked.

   $G((Vo)W(\Sigma_{n-1}^{n} Pit^{n}) \rightarrow n = k$

   Verify order must wait until pick items executes for a specified number of times i.e. until all items are picked where k = number of items.

6. Where stock of items is not available for an order, suspend order and contact customer

   $G\left(\Sigma_{n+1_n}^{n} Pit, F(Suspend \wedge Contactcustomer)\right)$

   If the items picked do not sum up to the items ordered (if no more items are available), the order is suspended, and the customer is contacted.

7. Unavailable items can be substituted upon permission from the customer
   $G\big(Pit \rightarrow [Item - unavailable], (Contactcustomer \wedge Replace) \vee F(alternativeitemsatdelivery)\big)$

   Where items on the order are not available, the customer is contacted to replace the items or alternative items are carried and offered during the order delivery.

8. The total order processing time is approximately 3 hours. The total duration for processing each case of the order is given by: Total process duration =
   $\sum_{t}(So, Pit, Vo, Po, Ho, Cpd)$

Using the formal specified verification requirements, the next section shows how to check for their fulfilment and compliance through application of the verification algorithms.

Application of Compliance Verification

To verify the business process's compliance with above constraints, the overall compliance verification algorithm 12 is applied. The specific properties verified in this case include the following:

Termination property: this property is used to check the possibility that a model has start and end points, i.e. a model can start and end. To check this property, the algorithm 12 checks for existence of initial and end activity events for each complete case in a process instance. Absence of initial and end events indicates lack of termination which is also a source of deadlocks i.e. tasks that start and never complete. It also violates the constraints for initial and end activities specified in requirement 1.

Deadlocks: checking for these deadlocks in models ensures that no activities remain stuck, incomplete or unexecuted due to lack of resources, resource overutilization or unintended lock out or denial to data access. For example, due to SoD restrictions, situations may arise where no resource is available to execute a task. The algorithm checks to detect deadlocks likely to be caused by resource allocation. This is enforced by checking constraints related to resource allocation to process activities such that deviant behaviour leading to violations can be detected early in time. From the use case, at least the supervisor role is assigned to each task as a continuity strategy. The algorithm further checks for the existence of roles that can free over allocated resources or execute tasks that may exist without assigned resources or whose resources may be busy. From the use case, the supervisor role is assigned for each task as specified in requirement 3, thus the algorithm checks for its existence. The non-existence of supervisor role assignment over tasks is considered a violation.

Livelocks: checking for livelock in the model ensures that no instances are trapped in infinite loops. For example, sources of livelocks in the use case are: orders that remain pending because of non-availability of stock items, orders that do not pass verification and executions that remain pending due to denied data access. Specification 7 allows item substitution where an ordered item is not available. 'is helps to prevent order suspension which is a likely source of livelocks. The algorithm in this case will verify for existence and permission to execute the substitute item activity in the model. Absence or lack of necessary resource assignments to execute this activity amounts to a violation.

Temporal conflicts checking: the verification of temporal constraints checks for conflicts related to temporal assignments where resources (roles) may be assigned to different tasks whose execution occurs at the same time, or activities that start and end at the same time yet assigned to same resource. This would imply that only one task may be attended to due to conflicts in execution time causing a delay in the entire process duration. The algorithm checks for conformance to temporal requirements and detecting likely deviations based on the total process duration.

Where the duration is beyond the total activity scheduled times, it implies a delay. The algorithm will proceed to check and identify the activities likely to cause delays and thus violating the temporal constraints. Requirement 8 specifies total order process instance duration to be 3 hours. The algorithm sums up the specific activity durations and delays to determine the compliancy to the required process cycle time. If the execution time exceeds the scheduled time, then a temporal violation is reported. Permission lock Property: the property relates to checking of conflicts relating to access control and authorizations where permissions may be granted and denied at the same time, or permit and authorize the same role for the same activity at the same time. 'is leads to permission locks which the algorithm assists to identify by assessing the data constraint assignments concerning access control and authorization, security and privacy.

From the case, access to data requires access and availability for the specific assigned roles except where customer data which is considered private as requirement 6 and 7 specify. Access to customer addresses is controlled by privacy constraint. The algorithm checks for compliance to this constraint.

To facilitate further evaluation of the artifacts’ outcomes, a practical implementation of a prototype is necessary.

Algorithm 12 Overall Compliance Verification - Pick and Pack Business Process

```
1: $InPut$:
       BP, Constraints
2: for all $Pi$:C.(e.ac) = init, end do
       init = So, end = Cpd
3:   if init $\in seen, finished \neq$ So then
       "Violation of Start activity constraint."
4:     if end $\in seen, finished \neq Cpd$ then
       "Violation of End activity constraint."
5:     end if
6:   end if
7: end for
8: Verify constraints BoD, SoD and Delegate
9: for all actors $r(Pickers, Packers, Verifiers, Supervisors) \in R$ do
       $(Pickers)$.BoD = $((So, Pit), Pickers)$
       $(Verifiers, Packers)$.SoD = $(Vo, Verifiers) \wedge (Vo', Packers), \neg Pickers)$
       $(Supervisor, Pickers)$.Delegate = $(Vo, (Verifiers \wedge Pickers))$
10:  if $(Packers).BoD \in seen, finished \neq ((So, Pit), Pickers)$ then
       "Violation of BoD constraint for Pickers over" $e.ac = (So, Pit)$
11:    if $(Verifiers, Packers).SoD \in seen, finished \neq (Vo, (Verifiers \wedge Packers))$ then
       "Violation of SoD constraint for Verifiers and Packers over Vo"
12:      if $(Supervisor, Pickers)$.Delegate $\in seen, finished \neq$(Vo,(Verifiers $\wedge Pickers)$) then
       "Violation of Delegate constraint for Supervisor and Pickers over Vo"
13:      end if
14:    end if
15:  end if
16: end for
17: Verify for existence of verify order for each $Pi$
18: for each $Pi$, C= Exist.Vo $\in \sigma$ do
       $\forall \sigma \in Pi \rightarrow \exists$ (Vo, Supervisor) =0
```

```
19:     if $\forall Pi \nexists$ Exist.Vo then
          "Constraint Violation for Exist.Vo"
20:         Verify for data constraint compliance $Pi$
21:       if Order = "Suspended" then
          $e.ac =$ Contact Customer, supervisor,[Read.customer data = authorise]
22:       else
          "Violation of data constraint, denied access to customer contact data"
23:       end if
24:     end if
25: end for
26: Verify compliance with Temporal constraints $Pi$
27: if Total $Pi$ Duration $\neq<$ $\sum$ e.ac $\in$Pi then
      "Violation: Instance delay detected"
28: end if
29: Feedback
      No Violation for start and end activity constraints for the provided business process.
      No Violation of BoD constraint for the provided business process.
      No Violation of SoD constraint for the provided business process.
      No Violation of Delegate constraint for the provided business process.
      No Violation of Existence of Verify order constraint for all instances in the process.
      No Violation of assignment of supervisor actor for all instances in the business
    process.
      No Violation of temporal constraint for the provided business process.
```

8. Conclusion

This deliverable provides an initial work for supporting end users to verify collaborative business processes for compliance with policy and regulatory requirements. The outcome is a compliancy verification approach supporting elicitation, specification and analysis of policy and regulatory requirements, their translation into formal constraints that are verifiable with collaborative business processes for compliance in the context of virtual factory.

In Chapter 2, we summarized process verification and compliance approaches of collaborative business processes based on our publications (Kasse, Xu and de Vrieze, 2017; Kasse *et al.*, 2018; Oyekola and Xu, 2020) and D1.3.

In Chapter 3, an example case and related to compliance issues and requirements are provided as a base to explain the motivation and the design of the approaches. The requirements are from different dimensions of processes and polices.

In Chapter 4, the different compliance constraints are expressed and specified using different formalizations.

In Chapter 5, the compliance verification is designed for verifying the control flow, resource flow and data flow respectively. Related definitions and algorithms are included.

In Chapter 6, process driven access control and authorisation are discussed for collabrative business processes. The example case is used for explaining the approaches.

In Chapter 7, the designed verification approaches are explained by using the example cased mentioned in Chapter 3.

The virtual factory will shift business processes from processes within one organization to collaborative business processes cross-organisations involving various partners, cutting across borders, and required to satisfy numerous policies, standards and regulations. This calls for stable, affordable yet usable supportive applicable tools, techniques and methods to support design and verification of collaborative business processes that are compliant to not only internal requirements but also external regulations. D4.1 presented a mechanism and algorithm to support the specification of data constraints and verifying for their compliancy with collaborative business processes.

The data constraint verification algorithm is designed based on an example business process case and evaluated with another example case. Besides, the algorithm's time performance requirements are also evaluated. To provide meaningful verification, feedback is provided on compliance or violation of relevant constraints. For future work, we target to integrate compliance verification for other process perspectives based on resource requirements in collaborative business processes. Moreover, a practical implementation prototype is of the algorithms forms our next step.

References

van der Aalst, W. M. P. (1997) ‘Verification of workflow nets’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, pp. 407–426. doi: 10.1007/3-540-63139-9_48.

van der Aalst, W. M. P. (2002) ‘Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques’, in *Business Process Management. Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, pp. 161–183. doi: 10.1007/3-540-45594-9_11.

van der Aalst, W. M. P. (2004) ‘Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management’, in Desel Jörgand Reisig, W. and R. G. (ed.) *Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 1–65. doi: 10.1007/978-3-540-27755-2_1.

van der Aalst, W. M. P., ter Hofstede, A. H. M. and Weske, M. (2003) ‘Business process management: A survey’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, pp. 1–12. doi: 10.1007/3-540-44895-0_1.

Awad, A., Weidlich, M. and Weske, M. (2009) ‘Specification, verification and explanation of violation for data aware compliance rules’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, Berlin, Heidelberg, pp. 500–515. doi: 10.1007/978-3-642-10383-4_37.

Borrego, D. and Barba, I. (2014) ‘Conformance checking and diagnosis for declarative business process models in data-aware scenarios’, *Expert Systems with Applications*, 41(11), pp. 5340–5352. doi: 10.1016/j.eswa.2014.03.010.

Corradini, F. *et al.* (2017) ‘BProVe: A formal verification framework for business process models’, in *ASE 2017 - Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. Institute of Electrical and Electronics Engineers Inc., pp. 217–228. doi: 10.1109/ASE.2017.8115635.

Corradini, F. *et al.* (2018) ‘Animating multiple instances in BPMN collaborations: From formal semantics to tool support’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, pp. 83–101. doi: 10.1007/978-3-319-98648-7_6.

Dwyer, M. B., Avrunin, G. S. and Corbett, J. C. (1998) ‘Property specification patterns for finite-state verification’, in *Proceedings of the Workshop on Formal Methods in Software Practice*. New York, New York, USA: ACM, pp. 7–15. doi: 10.1145/298595.298598.

Ferraiolo, D. F. *et al.* (2001) ‘Proposed NIST Standard for Role-Based Access Control’, *ACM Transactions on Information and System Security*, 4(3), pp. 224–274. doi: 10.1145/501978.501980.

Finkel, A. and Schnoebelen, P. (2001) ‘Well-structured transition systems everywhere!’, *Theoretical Computer Science*, 256(1–2), pp. 63–92. doi: 10.1016/S0304-3975(00)00102-X.

El Gammal, A. *et al.* (2016) ‘Formalizing and appling compliance patterns for business process compliance’, *Software and Systems Modeling*, 15(1), pp. 119–146. doi: 10.1007/s10270-014-0395-3.

El Gammal, A., Sebahi, S. and Turetken, O. (2014) *Business Process Compliance Management: An Integrated Proactive Approach*. International Business Information Management Association (IBIMA).

Goedertier, S. and Vanthienen, J. (2006) ‘Designing compliant business processes with obligations and permissions’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, pp. 5–14. doi:

10.1007/11837862_2.

Governatori, G. and Sadiq, S. (2009) ‘The Journey to Business Process Compliance’, in *Handbook of Research on Business Process Modeling*. IGI Global, pp. 426–454. doi: 10.4018/978-1-60566-288-6.ch020.

Groefsema, H. (2016) *Business Process Variability A Study into Process Management and Verification*. Rijksuniversiteit Groningen.

Houhou, S. *et al.* (2019) ‘A First-Order Logic Semantics for Communication-Parametric BPMN Collaborations’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, pp. 52–68. doi: 10.1007/978-3-030-26619-6_6.

Hu, V. C. *et al.* (2014) ‘Guide to attribute based access control (abac) definition and considerations’, *NIST Special Publication*, 800, p. 162. doi: 10.6028/NIST.SP.800-162.

Hu, V. C. *et al.* (2015) ‘Attribute-Based Access Control’, *Computer*, 48(2), pp. 85–88. doi: 10.1109/MC.2015.33.

Jin, X., Krishnan, R. and Sandhu, R. (2012) ‘A unified attribute-based access control model covering DAC, MAC and RBAC’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, Berlin, Heidelberg, pp. 41–55. doi: 10.1007/978-3-642-31540-4_4.

Kasse, J. P. *et al.* (2018) ‘The Need for Compliance Verification in Collaborative Business Processes’, in *IFIP Advances in Information and Communication Technology*, pp. 217–229. doi: 10.1007/978-3-319-99127-6_19.

Kasse, J. P. (2019) *Supporting Compliance Verification for Collaborative Business Processes*. Bournemouth University.

Kasse, J. P. *et al.* (2020) ‘Process Driven Access Control and Authorization Approach’, in *Advances in Intelligent Systems and Computing*. Springer, pp. 313–322. doi: 10.1007/978-981-15-0637-6_26.

Kasse, J. P., Xu, L. and de Vrieze, P. (2017) *A comparative assessment of collaborative business process verification approaches*, *IFIP Advances in Information and Communication Technology*. doi: 10.1007/978-3-319-65151-4_33.

Knuplesch, D. *et al.* (2013) ‘Towards Compliance of Cross-Organizational Processes and Their Changes’, in. Springer, Berlin, Heidelberg, pp. 649–661. doi: 10.1007/978-3-642-36285-9_65.

Knuplesch, D., Reichert, M. and Kumar, A. (2017) ‘A framework for visually monitoring business process compliance’, *Information Systems*, 64, pp. 381–409. doi: 10.1016/j.is.2016.10.006.

Oyekola, O. and Xu, L. (2020) ‘Verification and Compliance in Collaborative Processes’, in *PRO-VE 2020: 21th IFIP Working Conference on Virtual Enterprises*. Valencia, Spain: Springer.

Patterson, D. (2020) *Facebook data privacy scandal: A cheat sheet*, *TechRepublic*. Available at: https://www.techrepublic.com/article/facebook-data-privacy-scandal-a-cheat-sheet/ (Accessed: 5 October 2020).

Pesic, M. (2008) *Constraint-based workflow management systems: shifting control to users*. doi: Urn:nbn:nl:ui:25-638413.

Ramezani Taghiabadi, E. (2016) *Understanding Non-compliance*. Technische Universiteit Eindhoven.

Roa, J., Chiotti, O. J. A. and Villarreal, P. D. (2015) ‘Detection of Anti-Patterns in the Control Flow of Collaborative Business Processes’, in Roa, J., Chiotti, O. J. A., and Villarreal, P. D. (eds) *Simposio Argentino de Ingeniería de Software (ASSE 2015) - JAIIO 44*. Rosario.

Roa, J., Chiotti, O. and Villarreal, P. (2016) ‘Specification of behavioral anti-patterns for the verification of block-structured Collaborative Business Processes’, *Information and Software Technology*, 75, pp. 148–170. doi: 10.1016/j.infsof.2016.01.001.

Robol, M., Salnitri, M. and Giorgini, P. (2017) ‘Toward GDPR-compliant socio-technical systems: Modeling language and reasoning framework’, in *Lecture Notes in Business Information Processing*. Springer Verlag, pp. 236–250. doi: 10.1007/978-3-319-70241-4_16.

Von Rosing, M. *et al.* (2014) ‘Business process model and notation-BPMN’, in *The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM*. Elsevier Inc., pp. 429–453. doi: 10.1016/B978-0-12-799959-3.00021-5.

Russell, N. *et al.* (2004) *Workflow Data Patterns*. Brisbane.

Russell, N. *et al.* (2005) ‘Workflow data patterns: Identification, representation and tool support’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer, Berlin, Heidelberg, pp. 353–368. doi: 10.1007/11568322_23.

Sandhu, R. (1995) ‘Rationale for the RBAC96 family of access control models’, in *Proceedings of the ACM Workshop on Role-Based Access Control*. New York, New York, USA: ACM, pp. 1–8. doi: 10.1145/270152.270167.

Satariano, A. (2019) *Google Is Fined $57 Million Under Europe’s Data Privacy Law - The New York Times, The New York Times*. Available at: https://www.nytimes.com/2019/01/21/technology/google-europe-gdpr-fine.html (Accessed: 5 October 2020).

Taghiabadi, E. R. *et al.* (2014) ‘Compliance checking of data-aware and resource-aware compliance requirements’, in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Springer Verlag, pp. 237–257. doi: 10.1007/978-3-662-45563-0_14.

Thomas, R. K. and Sandhu, R. S. (1994) ‘Conceptual foundations for a model of task-based authorizations’, in *Proceedings of the Computer Security Foundations Workshop*. Franconia, NH: IEEE Computer Society, pp. 66–79. doi: 10.1109/CSFW.1994.315946.

Weske, M. (2007) *Business process management: Concepts, languages, architectures, Business Process Management: Concepts, Languages, Architectures*. Springer Berlin Heidelberg. doi: 10.1007/978-3-540-73522-9.

Wong, P. Y. H. and Gibbons, J. (2011) ‘Formalisations and applications of BPMN’, in *Science of Computer Programming*. Elsevier, pp. 633–650. doi: 10.1016/j.scico.2009.09.010.

Wynn, M. T. *et al.* (2009) ‘Business process verification - Finally a reality!’, *Business Process Management Journal*, 15(1), pp. 74–92. doi: 10.1108/14637150910931479.

Xu, L. (2003) ‘Monitorable electronic contract’, in *Proceedings - IEEE International Conference on E-Commerce, CEC 2003*. Institute of Electrical and Electronics Engineers Inc., pp. 92–99. doi: 10.1109/COEC.2003.1210238.

Xu, L. and Jeusfeld, M. A. (2003) ‘Pro-active monitoring of electronic contracts’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2681, pp. 584–600. doi: 10.1007/3-540-45017-3_39.