JOT
JOURNAL OF
OBJECT TECHNOLOGY

Journal of Object Technology | **RESEARCH ARTICLE**

Towards a Security Framework for Artifact-centric Workflows Leveraging Runtime Enforcement

Gaurav Gupta*, **Saumya Shankar***, **and Srinivas Pinisetty***
*Indian Institute of Technology Bhubaneswar, India

ABSTRACT A business process is made up of a set of activities that are carried out to create products or services. This series of activities is often analogous to the set of actions performed on an artifact (document). Aside from privacy (unauthorized access should be avoided) and integrity (the document should not be tampered with), the document has “lifecycle" constraints (modifications should be made in a predefined sequence). Any document manipulation that does not adhere to the lifecycle constraints is considered invalid. So far, cryptographic, centralized, and static verification approaches have been exploited to achieve compliance with the lifecycle, which have their respective limitations.

In this paper, we design and develop a framework, leveraging formal runtime enforcement approaches, to enforce the lifecycle constraints of a document at runtime, preserving its integrity and privacy using cryptographic approaches alongside. The lifecycle constraints are taken as the specification of the system, and the modification done to the document is taken as the (possibly erroneous) input to be enforced. The enforcement mechanism detects any modification attempt made by an individual into the document that is not following the set lifecycle constraints at runtime, and the document is safeguarded from such invalid manipulations. We take examples of a collaborative project between an academic and a research institute and a loan application process in the banking sector. We specify the necessary lifecycle constraints in these scenarios and construct enforcement monitors out of them, which will prevent any unauthorized changes to the document, assuring the system’s safety. The proposed framework has been implemented, and enforcement of constraints is demonstrated in the considered scenarios.

KEYWORDS Artifact Lifecycle, Runtime Enforcement, Cryptography, Business Process Models

1. Introduction

Among the business process models, artifact-centric workflow models allow us to define artifacts (data records) and their life cycles (evolution of data). These workflows are a series of activities performed in coordination, to produce a product or service. It takes into account both structural (i.e. the data) and dynamic (i.e. the activities or tasks) dimensions of the process. The series of activities in a business process can be mapped to possible actions that can be performed on artifacts/documents as it passes from one person to the other.

For example, considering the medical field, the document assigned to the patient, when he visits a hospital, is modified from person to person: first, the doctor prescribes drugs in the document; then, the pharmacist writes the cost of drugs; the insurance company writes approval of expenses into the document; and the nurse writes the timestamps, she gave the drug to the patient, etc.

Constraints on the document. However, sometimes there are certain constraints such as “integrity" constraints on the document, ensuring that the document passed is genuine and has not been manipulated/tampered with. For example, in the scenario considered above, one can have the following integrity constraints:

JOT reference format:
Gaurav Gupta, Saumya Shankar, and Srinivas Pinisetty. *Towards a Security Framework for Artifact-centric Workflows Leveraging Runtime Enforcement.* Journal of Object Technology. Vol. 23, No. 2, 2024.
http://dx.doi.org/10.5381/jot.2024.23.2.a1

“The pharmacist cannot write test results", or “The nurses cannot prescribe drugs". So, the concerned document (prescription note) will be discarded if unauthorized modifications are made to it.

Similarly, there are constraints to protect the “privacy" of the document. Privacy rights ensure that the peers have access to any information in our data/document and any unauthorized access is prohibited. For example, in the scenario considered above, one can have privacy constraints as “Insurance companies cannot access test results", or “Doctors cannot access insurance policy numbers". Thus, giving complete control over sensitive data.

Moreover, there are constraints on the “lifecycle" of the document which is, the sequence of manipulations performed in the document. By defining a lifecycle and complying with it, an organization can standardize its processes throughout and track each activity at every stage of the lifecycle, which eases the process and makes it less prone to errors. Under these constraints, all the manipulations not done in their prescribed order are considered invalid. For example, in the above scenario, if we consider a lifecycle constraint saying that, “The pharmacists cannot change/write the cost of drugs once it is approved by the insurance company", then, any out-of-order activity (pharmacists trying to access the document after it has been accessed by the insurance companies) is prohibited during enforcement of the lifecycle constraints.

Cryptographic approaches for ensuring integrity and privacy constraints. (Halle et al. 2016) discusses cryptographic ways for ensuring the integrity and privacy of the document. 1) Peers from one institution can be “grouped" together to establish privacy boundaries. 2) Sequence of data entered by a peer can be encrypted (to hide information from other groups), and its “digest" (explained in Sect. 3) is computed. This digest can be verified to detect any tampering with the document, thus ensuring its integrity. Sect. 3 delves into details of the cryptographic ways for ensuring integrity and privacy.

Runtime monitoring approaches for ensuring the lifecycle constraints. The lifecycle compliance can also be checked/ensured by cryptographic approaches. Hallé et al. in (Halle et al. 2016) do a series of modifications to the document which symbolized the lifecycle of the document. The “digest" is computed after each action (at runtime) and can be verified to detect any violation in the lifecycle compliance. Formal methods can also be employed to achieve the same. For example, static verification can be used (Gonzalez et al. 2012) to exhaustively search the state space of the design to carefully investigate any possible erroneous activity with respect to the lifecycle. However, it cannot prevent invalid behaviors from occurring at runtime. This paper proposes to use formal runtime monitoring approaches for checking lifecycle compliance.

Runtime monitoring includes Runtime Verification (RV) and Runtime Enforcement (RE) techniques. RV allows one to check if a run of a system under observation complies with (or violates) the specification. The RV monitor has the capacity to consume events generated by a running system and emit verdicts depending on the history of events it has previously received and the current satisfaction of the property. In most simple cases the verdict would be either true or false. However, many runtime verification systems use verdicts containing three or more values to give a more fine-grained result. Moreover, it is beneficial in some cases, to ensure that property violations are not only detected but prevented, i.e., not just detecting errors or violations, but also taking proactive measures to prevent them from occurring in the first place. By employing runtime enforcement approaches, we can increase the reliability and robustness of a system, leading to improved overall performance and reduced risk of failure.

One way to check lifecycle compliance leveraging formal runtime monitoring approaches is to schedule the offline runtime verification process as a *cron*[1] job to monitor the lifecycle compliance. The manipulations / system events are recorded as an execution trace and whenever a satisfactory number of manipulations have been done/recorded, the collected execution trace is forwarded to the offline monitor to check if the series of manipulations are performed according to the lifecycle or not. However, it would be preferable if the violations were checked concurrently during the execution. This necessitates validating the lifecycle constraints online (at runtime).

Contribution. In this paper, we design and develop a framework[2] to enforce the lifecycle constraints of a document at runtime using formal runtime monitoring approaches, alongside preserving its integrity and privacy using cryptographic approaches. Runtime Enforcement (RE) consists of checking and ensuring that a run of a system complies with the (formal) specification of the system. Here, the lifecycle constraints are taken as the specification and the sequence of modifications done to the document is taken as the trace to be verified against the specification of the system. Any modification done by a peer into the document, which is not complying with the lifecycle constraints is checked at runtime and the document is protected from such invalid manipulations.

To demonstrate the practicability of the proposed approach, we consider two different scenarios. We specify some lifecycle constraints in these scenarios and demonstrate the enforcement of these lifecycles constraints.

Outline. Sect. 2 gives the overview of the proposed framework. Sect. 3 discusses the cryptographic ways of complying with obligatory constraints such as integrity and privacy constraints. Sect. 4 discusses enforcing lifecycle constraints using runtime enforcement approaches. Sect. 5 presents implementation and discussions. Related literature is presented in Sect. 6 and Sect. 7 concludes the work.

[1] Cron is a service running in the background that will execute commands (jobs) at a specified time, or at a regular interval.

[2] The framework is implemented and is available for download at https://github.com/gg1711/Towards-a-Secure-Framework-for-Artifact-centric-Workflows-Leveraging-Runtime-Enforcement

2. Overview of the proposed framework

In this section, we give an overview of the proposed framework. In Fig. 1, we present the high-level framework, where Alice is the peer who attempts to modify a document $\mathcal{D}$ or wants to check the integrity of the document, to find out if it has been tampered with or not. M is the enforcement monitor (synthesized using RE approaches) composed out of the specifications (i.e., lifecycle to be enforced), and V is the verifier block which will look for tampering if any (synthesized using cryptographic approaches). Action $w(\delta)$ denotes writing δ into $\mathcal{D}$, v denotes the peer's will to verify the document, and r denotes reading the document.

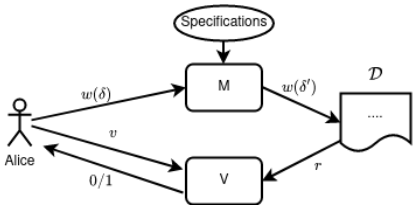

Figure 1 High level framework for verifying the constraints

In the framework, if a peer attempts to modify a document ($w(\delta)$), then the monitor checks if the peer's access is consistent with the lifecycle constraints. If it follows the constraints, then the modification is done ($w(\delta') = w(\delta)$), otherwise it is prevented ($w(\delta') = \phi$). Moreover, if a peer wants to verify the document (v), then the verifier block will read (r) the document $\mathcal{D}$ and check if any tampering is done to the document and thus, will give 0 (not tampered) or 1 (tampered) as a response.

To demonstrate the adaptable use of the proposed approach, we consider two scenarios: the first scenario is of sanctioning of a collaborative research project, and; the second scenario is of a loan application process in the banking sector.

Collaborative project: Consider the scenario of a collaborative research project between an academic and a research institute, as shown in Fig 2. Consider that the academic institute consists of peers such as *students*, *faculty*, and *research lead/advisor* and the research institute consists of peers such as *research lead/advisor*, *data scientist*, and *research director*. So whenever a joint project proposal is put forward then the supporting documentation is modified/approved from person to person: first, the *student* (from the academic institute) will write the project proposal and the project budget; then its feasibility is validated by his *faculty* in charge and the *research lead* of his institute; the *data scientist* (of the research institute) will analyze the project proposal and will offer better insights on the best solutions for the project within the budget, and then the document is sent to the *research director* for sanctioning of the project.

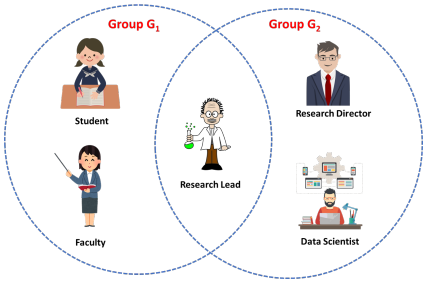

Figure 2 Scenario 1: Collaborative project

For illustration, we consider the following grouping of the peers and the lifecycle constraints (listed below) for the unified project: Peers *student* (A), *faculty* (B), and *research lead* (C) be grouped in group G_1 and peers *research lead* (C), *data scientist* (D), and *research director* (E) be grouped in group G_2. Thus, we have peers $P = \{A, B, C, D, E\}$ and two groups $G = \{G_1, G_2\}$ for academic and research institutes respectively, where $G_1 = \{A, B, C\}$ and $G_2 = \{C, D, E\}$.

1. A *student* can undertake a research project only after it is approved by the *faculty* and the *research lead*.

2. *Research director* can initiate the funding process only after the *data scientist* has approved.

3. *Faculties* can use data for experiments only after it has been prepared and released by the *data scientist*.

Loan application process: Consider the scenario of a loan application process in banking sectors, as shown in Fig 3. Let us consider peers *borrower*, *loan officer*, *underwriter*, *credit officer*, and *loan closer* involved in the loan sanctioning process. So, whenever a *borrower* wants a loan to be sanctioned then, the supporting documentation is modified/approved again from person to person: first, the *borrower* submits a loan application to the bank; then the *loan officer* will provide all information needed to prequalify the loan request and will refinance existing debt if required; the *underwriter* verifies and analyzes the submitted documents; then, it goes to the *credit officer* who ultimately approves or rejects the loan request, and will write the final decision i.e., approval or rejection (mentioning the reason behind the rejection) of the loan into the document; lastly the *loan closer* will review the documents and move the loan into the last phase of closing the loan.

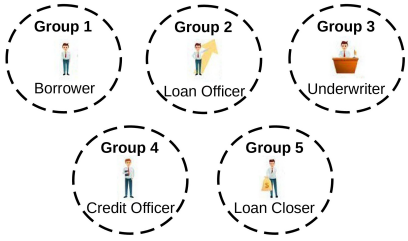

Figure 3 Scenario 2: Loan application process

For illustration, we consider the following grouping of the peers and the lifecycle constraints (listed below) for the sanctioning of the loan: Peer *borrower* (A) be grouped in group G_1, *loan officer* (B) be grouped in group G_2, *underwriter* (C) be grouped in group G_3, *credit officer* (D) be grouped in group G_4, and *loan closer* (E) be grouped in group G_5. Thus, we have the peers set $P = \{A, B, C, D, E\}$ and five groups $G = \{G_1, G_2, G_3, G_4, G_5\}$, where $G_1 = \{A\}$, $G_2 = \{B\}$, $G_3 = \{C\}$, $G_4 = \{D\}$, $G_5 = \{E\}$.

1. In case of rejection[3] of the loan, the *loan closer*[4] can only write if the *credit officer* has written twice[5] into the document, and any successive attempts by any of the peer is not allowed.

2. The *borrower* can only write once into the document, as changing the loan amount, mortgage, etc. in the loan application will have to restart the loan sanctioning process.

Note: The constraints are specified in plain language here for easy understanding and are later written into easy-rte policy format to be converted into respective monitors (see below sections 5.2 and 5.3).

Enforcing lifecycle constraints: These lifecycle constraints are taken as the specifications of the scenario, that we want to monitor, and are used to construct the enforcement monitor. So, whenever a system event (modification done to the document) occurs, the monitor will examine if it complies with the lifecycle constraints. If it complies, then the modification is done, otherwise, it is prevented by the monitor and the monitor waits for the next event.

We used approaches proposed in (Pearce et al. 2020)(Pinisetty et al. 2017b) to synthesize the enforcement monitor, as these approaches synthesize the monitoring code directly from the specification provided. The generated monitor from the specified approaches also ensures correctness. It uses Valued Discrete Timed Automaton (VDTA) (Pearce et al. 2020) to formally define the specifications. VDTA is an automaton with a finite set of locations and discrete clocks (clocks are used to represent the evolution of time). It also has external input-output channels for system data and internal variables for internal computations. The use of clocks to account for the time between events and the use of the input-output channels to carry data values to/from the monitored system along with the event helps make a step towards practical RE.

Enforcing integrity and privacy constraints: We also enforced the integrity and privacy constraints on the document using cryptographic approaches. Integrity constraints ensure that the document has not been tampered with. Following (Halle et al. 2016), we also compute digest from the sequence of data entered by a peer and check it to see if there is any evidence of tampering. Privacy constraints make certain that we have access to the information in the document and any unauthorized access is prohibited. To create privacy borders, peers are grouped as per our requirements. The flowchart in Fig. 4 depicts our entire strategy.

In the flowchart in Fig. 4, whenever a peer P_i of a group G_i comes forward to access a document[6], it is presented the option to first find out if the document is tampered with or not by verifying the digest. If the peer proceeds, the $verify\_digest$ function is invoked, which takes the document $\mathcal{D}$ and checks the same. If the document has been tampered with, the process exits (necessary actions are taken), otherwise the document is read ($read\_action$) by the peer after giving its details (group ID, peer ID, etc.). The peer is further asked its will to write into the document $\mathcal{D}$. If the peer wants to write into the document, then the data to be written is read and this action sequence is sent to the monitor which will verify if the action sequence is violating the lifecycle constraints or not. If it is not violating the constraints, the action is performed i.e., the document is fetched and updated. Otherwise, the action is suppressed/prevented and the process repeats. This way the modifications on the document always comply with the constraints. The flowchart is divided into two major sections/modules: the first module employing cryptographic approaches to comply with integrity and privacy constraints and the second module employing runtime enforcement (enclosed within dotted lines) to comply with the lifecycle constraints.

In the coming sections, we will discuss various approaches to enforce the needed constraints on the document.

3. Enforcing integrity and privacy constraints using cryptographic approaches

In this section, we first discuss different ways of enforcing integrity and privacy constraints e.g., centralized workflow approaches, decentralized workflow approaches, cryptographic approaches, etc. We then discuss the approach followed in our framework (i.e., cryptographic approaches), and the general steps followed in the considered approach.

To enforce the integrity and privacy constraints discussed in Sect. 2, one can adopt the following:

1. *Mutual trust:* One can rely on mutual trust, meaning each party trusts the other. Thus, anytime a document goes

[3] Currently in the paper only the constraint about "rejecting" the loan has been considered. One can have similar constraints about "accepting" the loan, and the user can choose any of the constraints as per the requirement.

[4] The *loan closer* would document the detailed reasons for the rejection in the loan file. This could include reasons such as insufficient income or assets, a low credit score, or a high debt-to-income ratio. The *loan closer* may also include any additional information or documentation that supports the decision to reject the loan. Additionally, the *loan closer* may need to notify the *borrower* of the loan rejection and provide them with a written explanation of why the loan was not approved. The *loan closer* may include this explanation in the loan document or provide it separately to the *borrower*. This writing of the reasons is important to ensure that all parties involved understand the reasons for the loan rejection and the next steps to take.

[5] We considered the scenario that, when a loan is rejected, the *credit officer* will do two things: 1.) write the annexure number of the rejection from the guidelines/rulebook and 2.) Signature to indicate that they have reviewed the application and made a decision. Thus, he will be writing twice into different sections of the document, in case of rejection of the loan.

[6] Note that the document can be accessed by any number of peer simultaneously.

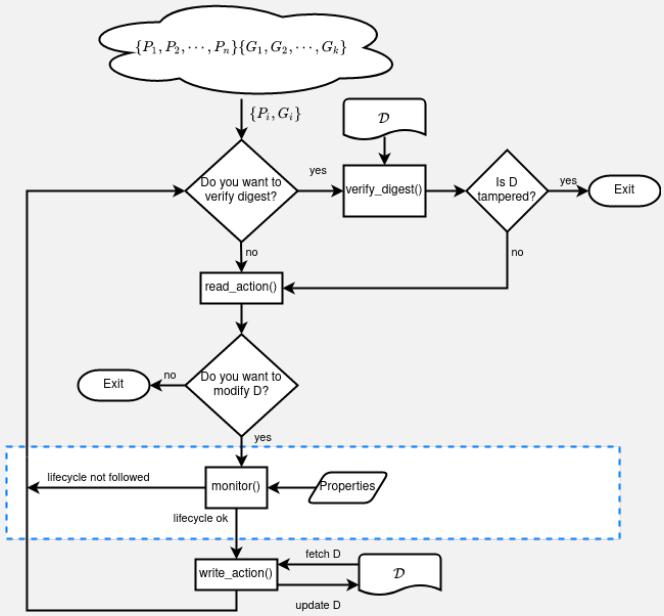

Figure 4 Process flow diagram

through a particular peer, they only access/change the part of the document they are permitted to change. This solution can only be used when the consequences of tampering with a document are not that severe, which is impossible in the real world. In the real world, we have so many business transactions through documents and we cannot blindly trust each other and assume that they will not forge or tamper with the document.

2. *Centralised workflow approaches:* These approaches consider all access and modifications controlled by the central server (Zhao et al. 2009). These systems always have a single point of failure, making them less reliable. Moreover, when multiple parties are involved in business artifacts, these systems are not trusted by everyone.

3. *Decentralised workflow approaches:* One can also check the enforcement of the lifecycle on documents at runtime and raise exceptions accordingly. Ouafi et al. (Ouafi and Vaudenay 2009) used this concept for the lifecycle enforcement of RFID tags. Runtime enforcement can also be employed for the same (discussed in Sect. 4). However, these approaches suffer from the problem of man-in-the-middle attacks as messages are not encrypted and the receiver cannot verify the sender's identity.

4. *Cryptographic approaches:* These approaches focus mainly on protecting the privacy and integrity of documents. The sequence of modifications done to the document is encrypted using cryptographic approaches which can only be decrypted by the receiver, thus, preventing the man-in-the-middle attacks. Document tampering can be detected by verifying the digest (explained below) computed from the sequence of data entered by the peer. The lifecycle enforcement, on the other hand, still rely on trust between the peers involved (Halle et al. 2016). Thus, limiting the overall functioning.

The below subsection discusses the cryptographic approaches to ensure the privacy and integrity of the document in our framework.

The cryptographic approach in our framework

In order to preserve the integrity of the document, metadata related to its history is attached to it, as discussed by Hallé et al. in (Halle et al. 2016). It allows any peer to retrieve the document, check the metadata, and verify if any tampering is done with it at any point in time.

Notations. The following notations are used throughout the paper: $E[M, K]$ denotes encryption of message M with key K. Similarly, $D[M, K]$ denotes decryption of message M with key K. $M[p, g]$ is the predicate which indicates if the peer p belongs to group g. We will also use the hash function $\hbar$ having output values in the set H. P is the set of peers and each peer $p \in P$ has a pair of public key $K_{p,u}$ and private key $K_{p,v}$. G is the set of groups based on the classical access control permissions. Peers can be members of more than one group. Each group member has access to its shared key S_g, which is shared only between group members.

The general approach (Halle et al. 2016) contains the following steps:

3.1. Encrypting a sequence

Each peer can take action on behalf of their group. Thus, peers encrypt actions using shared group key S_g as $E[a, S_g]$. Since non-group members do not have access to the group key, they

can only know that some action is performed on the document but cannot decrypt what "exact" action is taken. They can see the peer-action sequence $< E[a, S_g], p, g, h >$, which means peer p has performed an action $E[a, S_g]$ on behalf of the group g and its digest h is computed.

3.2. Computing the digest

To detect tampering with the document, we compute a digest which is calculated using the complete history of the peer-action sequences done from the documents' initial state. It is calculated using the below equation:

Let $\bar{s}^* = \left(pa_1^*, \ldots, pa_n^*\right)$ be an encrypted peer-action sequence of length n, and let $\overline{s'}^* = \left(pa_1^*, \ldots, pa_{n-1}^*\right)$ be the same sequence without its last peer-action pair, and $pa_i^* = \left\langle a_i^*, p_i, g_i, h_i \right\rangle$ for $i \in [0, n]$. The digest of $\bar{s}^*$, noted $\nabla\left(\bar{s}^*\right)$, is defined as follows:

$$\nabla\left(\bar{s}^*\right) \triangleq \begin{cases} 0 & \text{if } n = 0 \\ E\left[\hbar\left(\nabla\left(\overline{s'}^*\right) \cdot a_n^* \cdot g_n\right), K_{p_n,v}\right] & \text{otherwise} \end{cases} \tag{1}$$

To compute the current digest, we have peer p_n performing the last action a_n on behalf of group g_n. a_n^* is its encrypted action i.e., $E[a_n, Sg_n]$. We take the hash of a concatenated string of last digest (i.e., $\nabla\left(\overline{s'}^*\right)$), a_n^* and g_n. We encrypt this hash with the private key of peer $K_{p_n,v}$.

The hash function ensures that concatenated string is of fixed length always. Digest $\nabla\left(\bar{s}^*\right)$ includes details of peer name, group name, and encrypted action so that the peer (verifier) can detect any tampering with the action sequence.

3.3. Checking the digest

The document should contain data along with peer-action sequences (i.e., $< E[a, S_g], p, g, h >$). The peer can detect document tampering by verifying the digest using the peer name, group membership, and hashes for each peer-action sequence.

Given an encrypted peer-action sequence $\bar{s}^* = \left(pa_1^*, \ldots, pa_n^*\right)$ of length n and the computed digest d of last action. Let $\overline{s'}^* = \left(pa_1^*, \ldots, pa_{n-1}^*\right)$ be the same sequence without its last peer-action pair, and $pa_i^* = \left\langle a_i^*, p_i, g_i, h_i \right\rangle$ for $i \in [0, n]$ The sequence $\bar{s}^*$ verifies d if and only if $\nabla^{-1}\left(\bar{s}^*, d\right)$ where:

$$\nabla^{-1}\left(\bar{s}^*, d\right) \triangleq \begin{cases} M\left(p_n, g_n\right) \wedge \exists \left\langle h, a^*, g \right\rangle : & \text{if } n > 0 \\ D\left[d, K_{p_{n,u}}\right] = \hbar\left(h \cdot a^* \cdot g\right) & \\ \wedge a^* = a_n^* & \\ \wedge g = g_n & \\ \wedge h_n = \hbar\left(h \cdot a^* \cdot g\right) & \\ \wedge \nabla^{-1}\left(\overline{s'}, h\right) & \\ \top & \text{otherwise} \end{cases}$$

Verifier peer can verify that for any given peer-action, only peer p_n has executed the action and it belongs to group g_n.

3.4. Decrypting a sequence

Before starting the decryption of a sequence, the peer first verifies the digest to ensure that the document's integrity is not compromised. Depending upon its group membership, the peer can decrypt the sequence on behalf of its group. In the case where the peer is a member of the group, the last action is decrypted using the group key $\left(D[a_n^*, S_{g_n}]\right)$, and the performed action is included. Otherwise, when the peer does not belong to the group, the entire peer-action sequence is discarded[7].

Summary. Thus, we have handled preserving integrity and privacy constraints using this approach. 1) We have protected the integrity of the document with the help of the digest, calculated using the complete history of the document. If a peer tampers the document, it can be easily detected. 2) We have protected the document's privacy by dividing our peers into several groups. Although all peers can see that some action is performed, only group members can decrypt the actual action performed. Group division of peers will help to set read-write permissions accordingly. The next section discusses enforcing lifecycle constraints by runtime enforcement approaches.

4. Enforcing lifecycle constraints using runtime enforcement

As discussed in Sect. 2, in this work, we employ RE approaches to enforce the lifecycle constraints of a document at runtime. We take the lifecycle constraints as the specifications of the system which we want to enforce and synthesize an enforcement monitor out of it using approaches proposed in (Pearce et al. 2020)(Pinisetty et al. 2017b). As mentioned in Sect. 2, these approaches synthesize the monitoring code directly from the specification. The generated monitor is sound (the output must comply with the specification), transparent (a correct input should be unchanged), and instantaneous (the output should be produced immediately) and thus, ensures correctness and safe behaviour in systems. Also, they use VDTA for characterizing the specification which helps in writing better enforcement specifications, thus ensuring compatibility with the real-world cyber-physical and industrial systems.

VDTA for defining the lifecycle constraints. VDTA is an automaton with a finite set of locations, a finite set of discrete clocks, external input-output channels, and internal variables. Here, we omit the details of the formal syntax and semantics of VDTA (Pinisetty et al. 2017b). We discuss the syntax of VDTA briefly via example specifications from collaborative project and loan application process scenarios.

Example 1 (Collaborative project scenario) *We consider the collaborative project system to have as many (Boolean) input channels*[8] *as there are peers in all groups i.e.,* $|G_1| + |G_2|$*, required as per the considered constraints to identify the peer*

[7] Discard here means that, since the artifact is shared, the action that the member wanted to do was not allowed to be executed on it.

[8] Outputs channels are not required in this framework, since we consider a uni-directional enforcer here, where, as illustrated in Fig. 4, the enforcer (monitor) takes the action sequence as input and authorizes/approves it w.r.t. the lifecycle constraints.

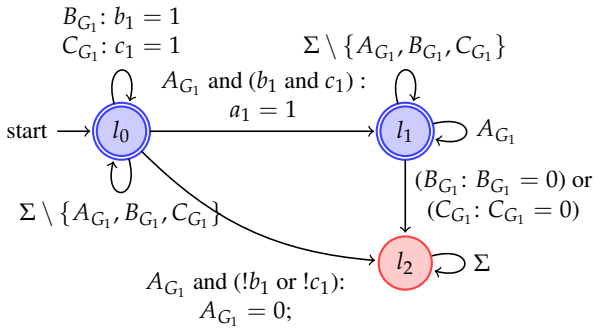

Figure 5 VDTA specifying the constraint: "*Peer A can undertake a research project only after it has been approved by peer {B, C}*".

performing an operation. We denote input channels by I. So, $I = \{A_{G_1}, B_{G_1}, C_{G_1}, C_{G_2}, D_{G_2}, E_{G_2}\}$, where each peer P_{G_i} denotes peer P is a member of group G_i. The alphabet Σ is $\Sigma = \Sigma_I = 2^I$. Also, each constraint makes use of a set of internal variables V where, $V = \{a_1, b_1, c_1, c_2, d_2, e_2\}$. These variables will be set to 1 whenever its corresponding peer has executed its modification. This will be used to track the order of modifications done by the peers. The first constraint, defined in Sect. 2, says that the student (A) can execute his modifications into the document after faculty (B) and research lead (C) have executed. We have modeled our constraint such that peer B or C should not perform an action after peer A has executed its action. It can be modeled as VDTA as shown in Fig. 5.

In Fig. 5, locations $\{l_0, l_1, l_2\}$ is the set of finite locations in which location l_0 is the initial location. Location $\{l_0, l_1\}$ are accepting locations and location l_2 is the violating location. According to the constraint, from initial location l_0 upon input B_{G_1} (indicating that peer B of group G_1 has taken the action) or C_{G_1}(indicating that peer C of group G_1 has taken the action), the VDTA remains at location l_0 and sets its internal variable b_1 and c_1 to 1 (true) respectively. Whereas, it goes to violating location l_2 on the reception of input A_{G_1} when b_1 or c_1 (or both) is 0 (false), indicating peer B_{G_1} or C_{G_1} (or both) has not performed their actions yet. It sets A_{G_1} of its input channel to 0, in this case, to discard this violating action.

Upon input A_{G_1} (indicating that peer A of group G_1 has taken the action) and when the Boolean internal variables b_1 and c_1 are 1 (true), the VDTA takes a transition to location l_1 and sets its internal variable a_1 to 1 (true). The VDTA remains in the same location on reception of input A_{G_1} and moves to the violating location l_2 on the reception of input B_{G_1} or C_{G_1} setting B_{G_1} or C_{G_1} of its input channel to 0 (false)[9].

Example 2 (Loan application process scenario) *Similar to the collaborative project system, the loan application system will have $|G_1| + |G_2| + |G_3| + |G_4| + |G_5|$ (Boolean) input channels. So, $I = \{A_{G_1}, B_{G_2}, C_{G_3}, D_{G_4}, E_{G_5}\}$. The alphabet $\Sigma = 2^I$. The first constraint, defined in Sect. 2, says that the loan closer (E) can only write/approve if the credit officer (D) has written twice into the document, in case of rejection of the loan. It can be modeled as VDTA as shown in Fig. 6 (top).*

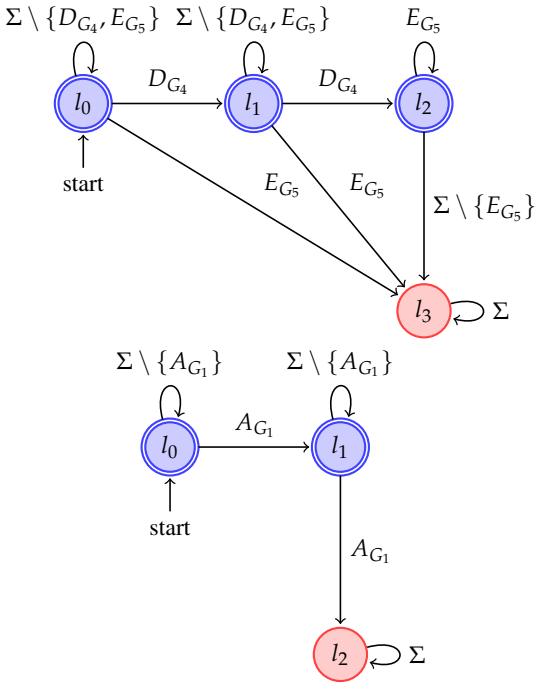

Figure 6 VDTA specifying the constraint: "*In case of rejection of the loan, peer E can only write/approve if peer D has written twice into the document*" (top), "*Peer A can only write once into the document*" (bottom).

In Fig. 6 (top), locations $\{l_0, l_1, l_2, l_3\}$ is the set of finite locations in which location l_0 is the initial location. Location $\{l_0, l_1, l_2\}$ are accepting locations and location l_3 is the violating location. According to the constraint, from initial location l_0 upon input D_{G_4}, the VDTA makes a transition to location l_1 (indicating peer D has modified once). Whereas, goes to violating location l_3 on reception of input E_{G_5}. It remains at the same location l_0 on reception of other inputs. From location l_1 upon input D_{G_4}, the VDTA makes a transition to location l_2 (indicating peer D has modified twice) and goes to violating location l_3 on reception of input E_{G_5}. It remains at the same location l_1 on reception of other inputs. On reception of input E_{G_5}, from location l_2, the VDTA remains at the same safe location and goes to the violating location l_3 when input $\Sigma \setminus \{E_{G_5}\}$ is received.

The same syntax works for Fig. 6 (bottom).

In Sect. 5, we will see the input traces getting enforced[10] according to the example constraints and will discuss the implementation details of the framework.

[9] We assume in constraint 1 that all the actions, performed by peers $\{B, C\}$ after peer A has executed his actions, are violating actions.

[10] We omit the details of the edit functions of VDTA framework to edit an erroneous input. It can be seen in (Pinisetty et al. 2017b).

5. Implementation and discussions

Implementation Structure: We have developed a Client-Server framework that will facilitate multiple clients to authenticate themselves and modify the document. The implementation is contained in two directories: *socket* and *erte* (as shown in Fig. 7).

The directory *socket* contains two Python files, server.py and client.py. The server.py implements the server socket (node) and client.py implements the client socket (node). First, the listener server socket listens on a particular port at an IP. It stores the list of registered peers (peer name and group name) in a dictionary.

```
registered_peers={'A':'G1',
                  'B':'G1',
                  'C':'G1',
                  'C':'G2',
                  'D':'G2',
                  'E':'G2'}
```

Then, the client socket tries to reach out to the server to form a connection. If the client is validated (by checking the IP address of the client and port number [11]), then the connection is established between the client and server, and they are ready to communicate: first, the client authenticates itself with peer name and group name, which the server matches with the dictionary containing stored registered peers; if the client is validated, then it is asked to enter an action to be executed.

The *erte* directory contains files required for implementing the cryptographic and RE approaches. The constraints/policies are specified in erte file contained in subdirectories inside the *Example* directory, which is converted[12] into an enforcer (written in C). The actions entered by the client are saved in actions.txt file (sample action.txt file is shown in Fig. 8). The Python file key_generation_peers.py contains the functions required to generate keys. The functions required to create a document, write into it, verify the digest, etc. are contained in Python file document_lifecycle.py.

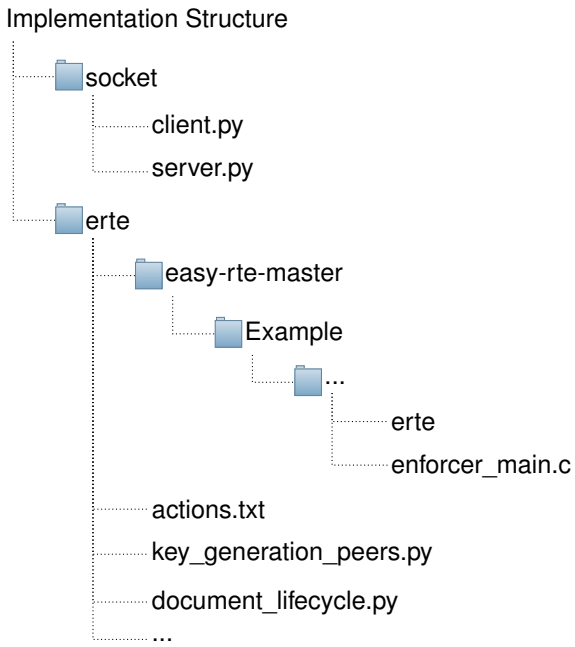

Figure 7 Implementation structure

5.1. Implementation

As our framework is divided into two modules (as depicted in the flowchart in Fig. 4), the implementation is also facilitated in the same way: the first module is associated with implementing the cryptographic approaches and the second module is associated with implementing the RE approaches.

Cryptography module. We have used the Python *RSA* library to generate the private-public keys (as shown in Fig. 9). These will be generated for each peer and group. Each peer can only access keys related to him (peer's personal key + groups key from which the peer belongs). The *hashlib* module in Python

```
D, G2, a(solution 2)
B, G1, e(budget)
A, G1, a(proposal)
C, G1, e(proposal)
A, G1, a(data 2)
A, G1, a(data 3)
B, G1, e(proposal)
```

```
A, G1, e(Borrower.amount=56000)
B, G2, a(loanofficer, refinance)
C, G3, e(Underwriter, verify)
D, G5, a(credit officer.sign)
B, G2, a(Loan officer.ok)
E, G4, e(loan.status=approved)
A, G1, a(interest)
```

Figure 8 Samples of write operations specified in action.txt files for collaborative project scenario (top) and loan application process scenario (bottom)

[11] We assume here simple validation and authentication approaches: the client is validated by an approach called "IP Address Filtering", where a server can validate the client's IP address against a whitelist of trusted IP addresses. However there can be various ways to validate/authenticate the client's identity. Here are a few common methods: Authentication, Digital Certificates, Challenge-Response Authentication, etc.

[12] The enforcer is automatically synthesized from the VDTA using the approaches in (Pearce et al. 2020)(Pinisetty et al. 2017b).

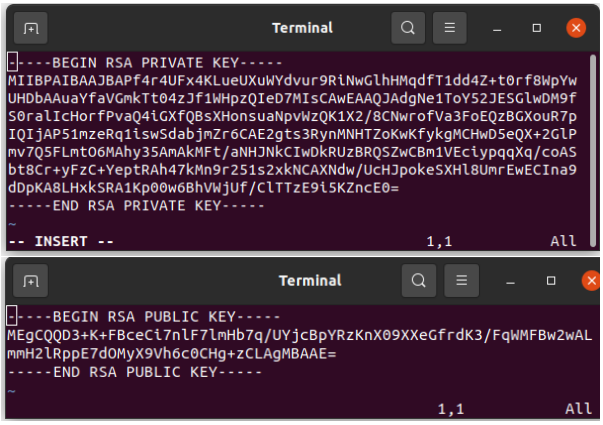

Figure 9 Generated sample private key (top) and public key (bottom) for collaborative project scenario

is utilized to calculate the digest, which uses the SHA-256 algorithm to hash data.

Peers can perform digest verification at any moment to detect any tampering with the document. If the document has been tampered with, it will be discarded and necessary actions will be taken. Peers can also perform read operations on behalf of a particular group into the document, to see the actions performed by the members of its group.

Runtime monitoring module. Peers can perform actions (write operations) on a shared document accessible to all the peers. Peers also need to specify "on behalf of" which group they are performing the action. The implementation will fetch the write action from the peer and will pass it to the monitor to detect any violation. If the action is validated by the monitor, the system will make the necessary modifications to the document as per the peer.

To detect violation of the constraints by an action performed by a peer, a monitor is constructed out of the specifications (lifecycle constraints) provided. It will take the system event and give an enforced output which will always satisfy the constraints. As discussed in Sect. 4, the followed approach uses VDTA for specifying the constraints which are written in their intended format.

The required peer-action sequence will be generated from the cryptographic module and stored in the shared document. A digest will be calculated based on the last action digest, group name, and the current action and the system will add this digest along with other necessary details to the action sequence.

5.2. Experimentation: Collaborative project scenario

We take the first constraint of the collaborative project scenario, defined in Sect. 2, modeled as VDTA as shown in Fig. 5 and listing 1, which says,

"The student (A) can execute his modifications into the document after faculty (B) and research lead (C) have executed".

According to the framework, any (write) action which does not obey the constraint is omitted (the modification is not performed in the document). Tab. 1 shows the incoming write actions into the document by the peers and the enforced actions.

Table 1 Incoming input and enforced input for the constraint 1 in Fig. 5

	S_I	M_I	M'_I	q	$EnfAct$	S_O
1	$w(D_{G_1},\delta)$	(0,0,0,0,1,0)	(0,0,0,0,1,0)	l_0	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$
2	$w(B_{G_1},\delta)$	(0,1,0,0,0,0)	(0,1,0,0,0,0)	l_0	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$
3	$w(A_{G_1},\delta)$	(1,0,0,0,0,0)	(0,0,0,0,0,0)	l_0	edtI	$\mathcal{D}_{new} = \mathcal{D}_{old}$
4	$w(C_{G_1},\delta)$	(0,0,1,0,0,0)	(0,0,1,0,0,0)	l_0	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$
5	$w(A_{G_1},\delta)$	(1,0,0,0,0,0)	(1,0,0,0,0,0)	l_1	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$
6	$w(A_{G_1},\delta)$	(1,0,0,0,0,0)	(1,0,0,0,0,0)	l_1	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$
7	$w(B_{G_1},\delta)$	(0,1,0,0,0,0)	(0,0,0,0,0,0)	l_1	edtI	$\mathcal{D}_{new} = \mathcal{D}_{old}$

```
interface of lifecycle_enforcer {
  in int16_t W_A_G1, W_B_G1, W_C_G1, W_C_G2,
    W_D_G2, W_E_G2;
  out int16_t is_p1_violated:=0;
  }
policy p1 of lifecycle_enforcer {
  internals{
        int16_t A_G1_p1 := 0 ;
        int16_t B_G1_p1 := 0 ;
        int16_t C_G1_p1 := 0 ;
       }
  states {
      s0 {
        -> s1 on (W_A_G1 and B_G1_p1=1 and
    C_G1_p1=1):
        A_G1_p1 := 1;
        -> s0 on (W_B_G1): B_G1_p1 := 1;
        -> s0 on (W_C_G1): C_G1_p1 := 1;
        -> violation on (W_A_G1 and
          (B_G1_p1=0 or C_G1_p1=0))
            recover is_p1_violated := 1;
        }
      s1 {
        -> s1 on (W_A_G1);
        -> violation on (W_B_G1 or W_C_G1)
            recover is_p1_violated := 1;
         }
       }}
```

Listing 1 VDTA specifying the constraint : "A can execute his modifications into the document after faculty B and C have executed".

In the table, S_I and S_O denote system input and output respectively and M_I and M'_I denote input coming to the monitor and input enforced by the monitor respectively. q denotes the location reached in the automaton (in Fig. 5) of constraint 1 and $EnfAct$ denotes the enforced action by the monitor on the input channel. fwdI denotes forwarding the input by the monitor when the input complies with the constraint; whereas edtI denotes editing the input by the monitor when the input does not comply with the constraint. $w(D_{G_1},\delta)$ under column S_I, denotes writing δ into the document by peer D of group G_1. It is the input received by the system from its peer. The tuples under column M_I hold the indexes of the peers from their respective groups who is carrying out the write action. It is the input forwarded by the system to the monitor. The mapping is done this way:

$(A_{G_1}:0, B_{G_1}:1, C_{G_1}:2, C_{G_2}:3, D_{G_2}:4, E_{G_2}:5)$. Thus, $(0,0,0,0,1,0) \equiv (A_{G_1}:0, B_{G_1}:0, C_{G_1}:0, C_{G_2}:0, D_{G_2}:1, E_{G_2}:0)$, for example, denotes peer D of group 2 carrying out the write action. Only a peer-group mapping index is required by the monitor to check the specified lifecycle constraints in our case. The tuples under M'_I denote the indexes of the peers from their respective groups whose action will be performed. It is the enforced input by the monitor to the system. For example, the enforced input $(0,0,0,0,0,0)$ against input $(1,0,0,0,0,0)$, under column M'_I in row 3 indicates that the corresponding peer is restrained from performing an action. Column S_O gives the modifications performed on the document.

In Tab. 1, initially the document was $\mathcal{D}$. Peer D performs the write operation and the monitor finds it satisfying the lifecycle constraints. Thus, the monitor does not modify the input channels ($M'_I = M_I$ and $EnfAct =$ fwdI) and the corresponding peer is allowed to perform its action. The document $\mathcal{D}$ ($\mathcal{D}_{old}$) is updated (i.e., $\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$) by appending peer-action sequence pa^*, which is $< E[\delta, S_2], D_{G_2}, G_2, \nabla(\bar{s}^*) >$ into it, as can be seen from the first row of Tab. 1. In the third row, when peer A performs a write operation into the document $\mathcal{D}_{old}$, the monitor finds the action violating constraint 1, thus, setting the index of the corresponding peer to 0 and enforcing ($EnfAct =$ edtI) the input $M'_I = (0,0,0,0,0,0)$ which indicates the action by the peer is restrained. It is because A has performed an action before B and C both have performed. Thus, it prevents this write operation into the document (document $\mathcal{D}$ remains unchanged i.e., $\mathcal{D}_{new} = \mathcal{D}_{old}$). This way the monitor intervenes every time and checks the violating modifications to the document to guarantee safe behaviour.

5.3. Experimentation: Loan application process scenario

We take both the constraints of the loan application process scenario, defined in Sect. 2, modeled as VDTA as shown in Fig. 6 and listing 2, which says,

"In case of rejection of the loan, the loan closer (E) can only write if the credit officer (D) has written twice into the document", and

"The borrower (A) can only write once into the document, as changing the loan amount, mortgage, etc. in the loan application will have to restart the loan sanctioning process".

According to the framework, any (write) action which does not obey both constraints is omitted (the modification is not performed in the document). Tab. 2 shows the incoming write actions into the document by the peers and the enforced actions.

```
interface of loan {
  in int16_t A, B, C, D, E;
  out int16_t is_p1_violated:=0;
  out int16_t is_p2_violated:=0;
}

policy p1 of loan {
  internals{
    int16_t d := 0 ;
  }
  states {
    s0 {
      -> s1 on D: d := 1;
    }
    s1 {
      -> s1 on D: d := 2;
      -> s1 on (E and d=2);
      -> violation on (E and d<2 ) recover
    is_p1_violated:=1;
       }
    }}


policy p2 of loan {
  states {
    s0 {
      -> s1 on A;
       }
    s1 {
      -> violation on A recover is_p2_violated
    :=1;
       }
       }}
```

Listing 2 VDTA specifying the constraints : "E can execute his modifications into the document after D have executed twice", and "A can only write once into the document".

Table 2 Incoming input and enforced input for the constraints in Fig. 6

	S_I	M_I	M'_I	q	$EnfAct$	S_O
1	$w(A_{G_1}, \delta)$	(1,0,0,0,0)	(1,0,0,0,0)	l_0l_1	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$
2	$w(B_{G_2}, \delta)$	(0,1,0,0,0)	(0,1,0,0,0)	l_0l_1	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$
3	$w(C_{G_3}, \delta)$	(0,0,1,0,0)	(0,0,1,0,0)	l_0l_1	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$
4	$w(D_{G_4}, \delta)$	(0,0,0,1,0)	(0,0,0,1,0)	l_1l_1	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$
5	$w(E_{G_5}, \delta)$	(0,0,0,0,1)	(0,0,0,0,0)	l_1l_1	edtI	$\mathcal{D}_{new} = \mathcal{D}_{old}$
6	$w(D_{G_4}, \delta)$	(0,0,0,1,0)	(0,0,0,1,0)	l_2l_1	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$
7	$w(A_{G_1}, \delta)$	(1,0,0,0,0)	(0,0,0,0,0)	l_2l_1	edtI	$\mathcal{D}_{new} = \mathcal{D}_{old}$
8	$w(E_{G_5}, \delta)$	(0,0,0,0,1)	(0,0,0,0,1)	l_2l_1	fwdI	$\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$

In table 2, q denotes the location reached in the product automaton of automata (in Fig. 6) of constraint 1. The mapping is done this way: $(A_{G_1}:0, B_{G_2}:1, C_{G_3}:2, , D_{G_4}:3, E_{G_5}:4)$. In Tab. 1, initially, the document was $\mathcal{D}$. Peer A performs the write operation and the monitor finds it satisfying the lifecycle constraints. Thus, the monitor does not modify the input channels ($M'_I = M_I$ and $EnfAct =$ fwdI) and the corresponding peer is allowed to perform its action. The document $\mathcal{D}$ ($\mathcal{D}_{old}$) is updated (i.e., $\mathcal{D}_{new} = \mathcal{D}_{old} + pa^*$) by appending peer-action sequence pa^*, which is $< E[\delta, S_1], A_{G_1}, G_1, \nabla(\bar{s}^*) >$ into it, as can be seen from the first row of Tab. 1. In the fifth row, when peer E performs a write operation into the document $\mathcal{D}_{old}$, the monitor finds the action violating constraint 1 (the product automaton of automata in Fig. 6, going to violating location l_3l_1), thus, setting the index of the corresponding peer to 0 and enforcing ($EnfAct =$ edtI) the input $M'_I = (0,0,0,0,0)$ which indicates the action by the peer is restrained. It is because

E has performed an action before D has written twice. Thus, it prevents this write operation into the document (document $\mathcal{D}$ remains unchanged i.e., $\mathcal{D}_{new} = \mathcal{D}_{old}$). Similarly, this way the monitor intervenes every time and checks the violating modifications to the document to guarantee safe behaviour.

The framework is implemented and is available for download at *https://github.com/gg1711/Towards-a-Secure-Framework-for-Artifact-centric-Workflows-Leveraging-Runtime-Enforcement*.

5.4. Discussions

Here, we discuss various points related to our framework and see how it can be utilized more efficiently.

1. *Scalability:* In our current approach, VDTA creates a product automaton of all the constraints together and traverses it (change its state), based on the input actions. This system works well and enforces the considered constraints within $10\mu s$ in our architecture,[13] which is reasonable. In the case of very complex (or more) constraints/specifications, one can avoid creating a product automaton and keep separate monitors for each constraint (Pinisetty and Tripakis 2016). In that case, the input action needs to be fed to each of the monitors simultaneously and if none of the constraints gets violated (all the monitors allow the input to go through), then that action can be performed, otherwise prevented. Thus, our framework can be scaled up efficiently even for substantially complex lifecycle constraints.

   For example, in Sect. 5.2, in order to enforce two constraints, first, the product automaton of automata in Fig. 6 (with the number of locations 4 and 3 respectively in their corresponding automaton) is computed. The final product automaton will have 12 locations, resulting in an increase in complexity. The further increase in the number of automata or the number of locations in an automaton of the constraints will increase the complexity of enforcement. However, compositional techniques can avoid this.

2. *Adaptability:* We only monitor the lifecycle constraints instead of the complete lifecycle, meaning, we specify what should not be done rather than specifying everything that peers should do in the document. This makes our framework more adaptable to many real-life scenarios. Also, writing constraints in our framework is relatively easier as the enforcement monitoring framework, based on VDTA, that we use provides many functionalities. The guards on internal variables, external variables and, clocks help to specify more realistic constraints. In this work, however, we have not realized its entire potential (to write more expressive policies). For example, complex constraints can be formulated, like, “A and B cannot write simultaneously", “A or B must write within 5 ticks after C or D have written", etc.

3. *Security:* The security strength of our framework depends upon the encryption-decryption algorithms used. If the artifact owner wants to use some other cryptographic algorithm other than RSA, they can easily do so by generating keys using that algorithm. Some state-of-the-art cryptographic algorithm includes e.g., Elliptic Curve Cryptography (ECC), EdDSA (Edwards-curve Digital Signature Algorithm), Diffie-Hellman key exchange, etc. The choice of the algorithm depends on various factors such as the level of security required, the size of the key, the speed of the algorithm, and the computing resources available.

4. *Decentralisation:* Any peer can check the tampering of the document by verifying the digest. Multiple peers can also read simultaneously into the document. Any peer can perform (write) actions in the document if it does not violate any of the lifecycle constraints. Currently, we have one monitor to verify the actions (and enforce them). In our future work, we plan to explore the possibility of using more than one instance of the same monitor simultaneously or having a decentralized migration algorithm (El-Hokayem and Falcone 2017) (where the migration algorithm will assign a monitor per component and the information is passed throughout the components to eventually verify the specification).

[13] Our architecture/machine is Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz with Quadro RTX 4000 and 8GB Graphics RAM / GPU.

6. Related Work

Artifact-centric business process. Over the last decade, business process modeling utilizing an artifact-centric approach has gained a lot of attention. Combining both data and process aspects makes it a valuable tool for effective business processing modeling. Nigam et al. (Nigam and Caswell 2003) introduced the concept of artifact-centric approaches related to workflow modeling. IBM tested his approach internally and found it robust and simple for workflow management. Later, a new set of requirements evolved that their procedural artifact-centric model (Nigam and Caswell 2003) could not solve. There are many frameworks that were proposed following the four-dimensional framework. Business Artifacts, Lifecycles, Services, and Associations (BALSA) is a four-dimensional framework that describes the artifact-centric modeling methodology (Bhattacharya et al. 2009) (Hull 2008). Multiple artifact-centric business process models with various characteristics can be generated by altering the model and constructs involved in each dimension. ArtiNets (Kucukoguz and Su 2010), ACP-i (Yongchareon et al. 2011), AXML (Abiteboul et al. 2009), GSM (Hull et al. 2010), and BPMN Extensions (Lohmann and Nyolt 2011) are examples of concrete artifact-centric modeling approaches.

Enforcement of artifact lifecycle by centralized and static verification approaches. There are various ways of enforcing a lifecycle on the artifacts of a business process. For example, Zhao et al. proposed, in (Zhao et al. 2009), a centralized approach for constructing business processes that included declarative and procedural methods. Ataullah et al. (Ataullah and Tompa 2011) proposed a method for converting finite state machine-based business policies into database triggers. In this approach, any modification in business objects result in multiple

transactions being done on a (central) database. There are works to statically analyze the lifecycle constraints. In (Bhattacharya et al. 2007), the authors present a formal model for artifact-centric business processes and discuss the static analysis of different practical concerns. Gerede et al. (Gerede and Su 2007) propose ABSL specification language for artifact lifecycle behaviours based on computation tree logic. Determining when the verification problem is decidable is also a study area. (Calvanese et al. 2014) identifies sufficient conditions which ensure that generated UML-based methodology models can be verified.

Runtime Monitoring. Artifact-centric business process. Over the last decade, business process modeling utilizing an artifact-centric approach has gained a lot of attention. Combining both data and process aspects makes it a valuable tool for effective business processing modeling. Nigam et al. (Nigam and Caswell 2003) introduced the concept of artifact-centric approaches related to workflow modeling. IBM tested his approach internally and found it robust and simple for workflow management. Later, a new set of requirements evolved that their procedural artifact-centric model (Nigam and Caswell 2003) could not solve. There are many frameworks that were proposed following the four-dimensional framework. Business Artifacts, Lifecycles, Services, and Associations (BALSA) is a four-dimensional framework that describes the artifact-centric modeling methodology (Bhattacharya et al. 2009) (Hull 2008). Multiple artifact-centric business process models with various characteristics can be generated by altering the model and constructs involved in each dimension. ArtiNets (Kucukoguz and Su 2010), ACP-i (Yongchareon et al. 2011), AXML (Abiteboul et al. 2009), GSM (Hull et al. 2010), and BPMN Extensions (Lohmann and Nyolt 2011) are examples of concrete artifact-centric modeling approaches.

Enforcement of artifact lifecycle by centralized and static verification approaches. There are various ways of enforcing a lifecycle on the artifacts of a business process. For example, Zhao et al. proposed, in (Zhao et al. 2009), a centralized approach for constructing business processes that included declarative and procedural methods. Ataullah et al. (Ataullah and Tompa 2011) proposed a method for converting finite state machine-based business policies into database triggers. In this approach, any modification in business objects results in multiple transactions being done on a (central) database. There are works to statically analyze the lifecycle constraints. In (Bhattacharya et al. 2007), the authors present a formal model for artifact-centric business processes and discuss the static analysis of different practical concerns. Gerede et al. (Gerede and Su 2007) propose ABSL specification language for artifact lifecycle behaviours based on computation tree logic. Determining when the verification problem is decidable is also a study area. (Calvanese et al. 2014) identifies sufficient conditions which ensure that generated UML-based methodology models can be verified. Runtime Monitoring. Runtime monitoring encompasses lightweight and dynamic (as compared with static monitoring approaches), verification and enforcement techniques that allow one to check (or enforce) if a run of a system under observation complies with (or violates) the specification. The specifications can be written using temporal logic (Bauer et al. 2011) or using high-level formalisms such as automata (Pinisetty et al. 2017a)(Pinisetty et al. 2014b). It employs a monitor which checks (one) current execution/trace of the system, thus a formal model of the system is not required and the system can be considered as a black box. Moreover, since it scales with the trace rather than the system model, issues like state explosion are avoided. The Runtime Verification (RV) monitor only examines if the system's actual execution is meeting or violating the specifications, whereas the RE monitor prevents the violation if it occurs by performing certain evasive actions. Blocking the execution, changing the input sequence by suppressing and/or adding events, and buffering input events until a future time when they could be forwarded are all examples of evasive actions.

In many systems, when an occasional malfunction happens there is a need to react instantaneously. The reaction can be shutting the system down or editing the system requests to ensure that the combined system complies with the specification. This is when RE (Pinisetty et al. 2013)(Pinisetty et al. 2014a) is needed. RE is an extension of RV aiming to circumvent property violations. Various enforcement frameworks differ from one another based on the enforcement mechanism (evasive actions) it employs and the language it uses to model the specifications of the system. (Schneider 2000) employs security automaton, a variant of Buchi automaton, to specify the security policies which are enforceable with mechanisms that work by monitoring system execution. (Ligatti et al. 2005)(Ligatti et al. 2009) employ edit automaton to specify the policies enforceable by monitoring the runtime behaviors of programs. An edit automaton combines the powers of suppression and insertion automaton. It can truncate action sequences and insert or suppress security-relevant actions at will. (Dolzhenko et al. 2015) employs mandatory results automaton which is obligated to return a result to the target application before seeing the next action it wishes to execute. Works in (Pearce et al. 2020)(Pinisetty et al. 2017b) deal with RE in reactive systems and cyber-physical systems. They allow altering the input events and use VDTA to specify the policies.

Enforcement of artifact lifecycle by runtime monitoring. (Halle et al. 2016) discusses some approaches (Bauer and Falcone 2011) (Colombo and Falcone 2016) which can reuse concepts from decentralized runtime monitoring. Checking if a run of a given system satisfies the formal specification of the system is termed runtime monitoring. These approaches do monitoring via progressing LTL- i.e., the monitor rewrites the formula to account for the new modifications starting with the LTL specification which makes the LTL formula very long. (Halle et al. 2016) further adds the limitations of these approaches. When document storage space is limited and the length of the sequence is long, it becomes difficult to store the new formula. (Halle et al. 2016) enforces lifecycle where the lifecycle is taken as the specification and the sequence of document modifications is taken as the trace to be verified. However, they do not implement policy-based runtime enforcement. The below comparison table 3 summarizes the notable works to enforce lifecycle constraints with their contributions and limitations. - The work by (Zhao et al. 2009), considers all access and modifications controlled

by the central server. Thus, the functionalities required to enforce lifecycle constraints can be implemented directly at this central location. However, these systems always have a single point of failure, making them less reliable. Moreover, when multiple parties are involved in business artifacts, these systems are not trusted by everyone. - The works by (Bhattacharya et al. 2007) and (Gonzalez et al. 2012) present a formal model for artifact-centric business processes and discuss the static analysis of different practical concerns. Static verification can be used to exhaustively search the state space of the design to carefully investigate any possible erroneous activity with respect to the lifecycle. However, it cannot prevent invalid behaviors from occurring at runtime. The works by (Ouafi and Vaudenay 2009) used decentralized workflow approaches to check the enforcement of the lifecycle on documents at runtime and raise exceptions accordingly. They used this concept for the lifecycle enforcement of RFID tags. However, these approaches suffer from the problem of man-in-the-middle attacks as messages are not encrypted and the receiver cannot verify the sender's identity. The works by (Halle et al. 2016) used digest computation to ensure enforcement of artifact lifecycles. They do a series of modifications to the document which symbolized the lifecycle of the document. The "digest" is computed after each action (at runtime) and can be verified to detect any violation in the lifecycle compliance. However, it assumes that the series of modifications are done according to the order. Our framework employs RE to develop a framework for enforcing the lifecycle constraints on the document. We use an automata-based runtime enforcement approach. We successfully enforce the lifecycle "constraints" into the document and ensure safe behaviour of the system. Runtime monitoring encompasses, lightweight and dynamic (as compared with static monitoring approaches), verification and enforcement techniques that allows one to check (or enforce) if a run of a system under observation complies with (or violates) the specification. The specifications can be written using temporal logic (Bauer et al. 2011) or using high-level formalisms such as automata (Pinisetty et al. 2017a)(Pinisetty et al. 2014b). It employs a monitor which checks (one) current execution/trace of the system, thus a formal model of the system is not required and the system can be considered as a black box. Moreover, since it scales with the trace rather than the system model, so issues like state explosion are avoided. The Runtime Verification (RV) monitor only examines if the system's actual execution is meeting or violating the specifications, whereas the RE monitor prevents the violation if it occurs by performing certain evasive actions. Blocking the execution, changing the input sequence by suppressing and/or adding events, and buffering input events until a future time when they could be forwarded are all examples of evasive actions.

In many systems, when an occasional malfunction happens there is a need to react instantaneously. The reaction can be shutting the system down or editing the system requests to ensure that the combined system complies with the specification. This is when RE (Pinisetty et al. 2013)(Pinisetty et al. 2014a) is needed. RE is an extension of RV aiming to circumvent property violations.

Various enforcement frameworks differ from one another based on the enforcement mechanism (evasive actions) it employs and the language it uses to model the specifications of the system. (Schneider 2000) employs security automaton, a variant of Buchi automaton, to specify the security policies which are enforceable with mechanisms that work by monitoring system execution. (Ligatti et al. 2005)(Ligatti et al. 2009) employ edit automaton to specify the policies enforceable by monitoring the runtime behaviors of programs. An edit automaton combines the powers of suppression and insertion automaton. It can truncate action sequences and insert or suppress security-relevant actions at will. (Dolzhenko et al. 2015) employs mandatory results automaton which are obligated to return a result to the target application before seeing the next action it wishes to execute. Works in (Pearce et al. 2020)(Pinisetty et al. 2017b) deals with RE in reactive systems and cyber-physical systems. They allow altering the input events and use VDTA to specify the policies.

Enforcement of artifact lifecycle by runtime monitoring. (Halle et al. 2016) discusses some approaches (Bauer and Falcone 2011) (Colombo and Falcone 2016) which can reuse concepts from decentralized runtime monitoring. Checking if a run of a given system satisfies the formal specification of the system is termed runtime monitoring. These approaches does monitoring via progressing LTL- i.e., the monitor rewrites the formula to account for the new modifications starting with the LTL specification which makes the LTL formula very long. (Halle et al. 2016) further adds the limitations of these approaches. When document storage space is limited and the length of the sequence is long, it becomes difficult to store the new formula. (Halle et al. 2016) enforces lifecycle where the lifecycle is taken as the specification and the sequence of document modifications is taken as the trace to be verified. However, they do not implement policy-based runtime enforcement.

The below comparison table 3 summarizes the notable works to enforce lifecycle constraints with their contributions and limitations.

- The work by (Zhao et al. 2009), considers all access and modifications controlled by the central server. Thus, the functionalities required to enforce lifecycle constraints can be implemented directly at this central location. However, these systems always have a single point of failure, making them less reliable. Moreover, when multiple parties are involved in business artifacts, these systems are not trusted by everyone.
- The works by (Bhattacharya et al. 2007) and (Gonzalez et al. 2012) present a formal model for artifact-centric business processes and discuss the static analysis of different practical concerns. Static verification can be used to exhaustively search the state space of the design to carefully investigate any possible erroneous activity with respect to the lifecycle. However, it cannot prevent invalid behaviors from occurring at runtime.
- The works by (Ouafi and Vaudenay 2009) used decentralized workflow approaches to check the enforcement of lifecycle on documents at runtime and raise exceptions

Work	Contribution	Limitations
(Zhao et al. 2009)	Central server	Single point of failure, less reliable.
(Gonzalez et al. 2012) and (Bhattacharya et al. 2007)	Static verification	Cannot prevent invalid behaviors from occurring at runtime
(Ouafi and Vaudenay 2009)	Decentralized approaches	Man-in-the-middle attacks
(Halle et al. 2016)	Digest computation	Assumption on the series of modification is done according to the order.

Table 3 Comparison table with similar approaches.

accordingly. They used this concept for the lifecycle enforcement of RFID tags. However, these approaches suffer from the problem of man-in-the-middle attacks as messages are not encrypted and the receiver cannot verify the sender's identity.

- The works by (Halle et al. 2016) used digest computation to ensure enforcement of artifact lifecycles. They do a series of modifications to the document which symbolized the lifecycle of the document. The "digest" is computed after each action (at runtime) and can be verified to detect any violation in the lifecycle compliance. However, it assumes that the series of modifications are done according to the order.

Our framework employs RE to develop a framework for enforcing the lifecycle constraints on the document. We use an automata-based runtime enforcement approach. We successfully enforce the lifecycle "constraints" into the document and ensure the safe behaviour of the system.

7. Conclusion

This paper presents a complete framework for enforcing the lifecycle constraints on the document while maintaining its privacy and integrity by employing runtime enforcement and cryptographic approaches. In this approach, the lifecycle constraints are specified and are taken as the specification of the system. The monitor is constructed from it. The modification done to the document is taken as the input to be verified. Any modification done by a peer on the document is according to the preferred order or not is checked at runtime. This way the document is protected from any invalid manipulation.

We considered scenarios of a collaborative project between an academic and a research institute and loan application process in banking sectors. We specified lifecycle constraints in these scenarios that should be followed while modifying the exchanging document. We constructed respective monitors out of these constraints which will prevent any violating modification to the document, thus ensuring safe behaviour in these systems. We implemented the framework and demonstrated the enforcement of constraints in the considered scenarios.

Acknowledgments

We would like to thank the reviewers and editors for their helpful comments and contributions. This work has been partially supported by The Ministry of Human Resource Development, Government of India (SPARC P#701), IIT Bhubaneswar Seed Grant (SP093).

References

Serge Abiteboul, Pierre Bourhis, Alban Galland, and Bogdan Marinoiu. The axml artifact model. In *2009 16th International Symposium on Temporal Representation and Reasoning*, pages 11–17, 2009. doi: 10.1109/TIME.2009.9. URL https://ieeexplore.ieee.org/document/5368565.

Ahmed Ataullah and Frank Tompa. Business policy modeling and enforcement in databases. *PVLDB*, 4:921–931, 08 2011. doi: 10.14778/3402707.3402730. URL https://dl.acm.org/doi/10.14778/3402707.3402730.

Andreas Bauer and Ylies Falcone. Decentralised ltl monitoring, 2011. URL https://link.springer.com/chapter/10.1007/978-3-642-32759-9_10.

Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime verification for ltl and tltl. *ACM Trans. Softw. Eng. Methodol.*, 20:14, 09 2011. doi: 10.1145/2000799.2000800. URL https://dl.acm.org/doi/10.1145/2000799.2000800.

Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. volume 4714, pages 288–304, 09 2007. ISBN 978-3-540-75182-3. URL https://link.springer.com/chapter/10.1007/978-3-540-75183-0_21.

Kamal Bhattacharya, Richard Hull, and Jianwen Su. *A Data-Centric Design Methodology for Business Processes*. 01 2009. doi: 10.4018/9781605662886.ch023. URL https://sites.cs.ucsb.edu/~su/papers/2008/BPM-handbook-chapter.pdf.

Diego Calvanese, Marco Montali, Montserrat Estañol, and Ernest Teniente. Verifiable UML artifact-centric business process models. In *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*. ACM, nov 2014. URL https://doi.org/10.1145%2F2661829.2662050.

Christian Colombo and Ylies Falcone. Organising ltl moni-

tors over distributed systems with a global clock. *Formal Methods in System Design*, 49, 10 2016. doi: 10.1007/s10703-016-0251-x. URL https://link.springer.com/chapter/10.1007/978-3-319-11164-3_12.

Egor Dolzhenko, Jay Ligatti, and Srikar Reddy. Modeling runtime enforcement with mandatory results automata. *Int. J. Inf. Sec.*, 14(1):47–60, 2015. URL https://doi.org/10.1007/s10207-014-0239-8.

Antoine El-Hokayem and Yliès Falcone. Themis: A tool for decentralized monitoring algorithms. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2017, page 372–375, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350761. doi: 10.1145/3092703.3098224. URL https://doi.org/10.1145/3092703.3098224.

Cagdas Gerede and Jianwen Su. Specification and verification of artifact behaviors in business process models. pages 181–192, 09 2007. ISBN 978-3-540-74973-8. URL https://link.springer.com/chapter/10.1007/978-3-540-74974-5_15.

Pavel Gonzalez, Andreas Griesmayer, and Alessio Lomuscio. Verifying gsm-based business artifacts. In *2012 IEEE 19th International Conference on Web Services*, pages 25–32, 2012. doi: 10.1109/ICWS.2012.31. URL https://ieeexplore.ieee.org/document/6257786.

Sylvain Halle, Raphael Khoury, Antoine El-Hokayem, and Ylies Falcone. Decentralized enforcement of artifact lifecycles. In *2016 IEEE 20th International Enterprise Distributed Object Computing Conference (EDOC)*, pages 1–10, Sep. 2016. doi: 10.1109/EDOC.2016.7579380. URL https://ieeexplore.ieee.org/document/7579380.

Richard Hull. Artifact-centric business process models: Brief survey of research results and challenges. volume 5332, pages 1152–1163, 11 2008. ISBN 978-3-540-88872-7. URL https://link.springer.com/chapter/10.1007/978-3-540-88873-4_17#citeas.

Richard Hull, Elio Damaggio, Fabiana Fournier, Manmohan Gupta, Fenno Heath, Stacy Hobson, Mark Linehan, Sridhar Maradugu, Anil Nigam, Noi Sukaviriya, and Roman Vaculin. Introducing the guard-stage-milestone approach for specifying business entity lifecycles. volume 6551, pages 1–24, 09 2010. URL https://link.springer.com/chapter/10.1007/978-3-642-19589-1_1.

Esra Kucukoguz and Jianwen Su. On lifecycle constraints of artifact-centric workflows. volume 6551, pages 71–85, 09 2010. ISBN 978-3-642-19588-4. URL https://link.springer.com/chapter/10.1007/978-3-642-19589-1_5.

Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4:2–16, 02 2005. doi: 10.1007/s10207-004-0046-8. URL https://link.springer.com/article/10.1007/s10207-004-0046-8.

Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, jan 2009. ISSN 1094-9224. URL https://doi.org/10.1145/1455526.1455532.

Niels Lohmann and Martin Nyolt. Artifact-centric modeling using bpmn. In *ICSOC Workshops*, 2011. URL https://link.springer.com/chapter/10.1007/978-3-642-31875-7_7.

A. Nigam and N. S. Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003. doi: 10.1147/sj.423.0428. URL https://ieeexplore.ieee.org/document/5386806.

Khaled Ouafi and Serge Vaudenay. Pathchecker: An rfid application for tracing products in supply-chains. Technical report, 2009.

Hammond Pearce, Srinivas Pinisetty, Partha S. Roop, Matthew M. Y. Kuo, and Abhisek Ukil. Smart i/o modules for mitigating cyber-physical attacks on industrial control systems. *IEEE Transactions on Industrial Informatics*, 16(7):4659–4669, 2020. doi: 10.1109/TII.2019.2945520. URL https://ieeexplore.ieee.org/document/8859335.

Srinivas Pinisetty and Stavros Tripakis. Compositional runtime enforcement. pages 82–99, 06 2016. ISBN 978-3-319-40647-3. doi: 10.1007/978-3-319-40648-0_7. URL https://dl.acm.org/doi/10.1007/978-3-319-40648-0_7.

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Landry Nguena Timo. Runtime enforcement of timed properties. In Shaz Qadeer and Serdar Tasiran, editors, *Runtime Verification*, pages 229–244. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-35632-2. URL https://link.springer.com/chapter/10.1007/978-3-642-35632-2_23.

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, and Hervé Marchand. Runtime enforcement of regular timed properties. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, page 1279–1286, New York, NY, USA, 2014a. ISBN 9781450324694. URL https://doi.org/10.1145/2554850.2554967.

Srinivas Pinisetty, Yliès Falcone, Thierry Jéron, Hervé Marchand, Antoine Rollet, and Omer Nguena Timo. Runtime enforcement of timed properties revisited. *Form. Methods Syst. Des.*, 45(3):381–422, dec 2014b. ISSN 0925-9856. URL https://doi.org/10.1007/s10703-014-0215-y.

Srinivas Pinisetty, Viorel Preoteasa, Stavros Tripakis, Thierry Jéron, Yliès Falcone, and Hervé Marchand. Predictive runtime enforcement. *Formal Methods in System Design*, 51, 08 2017a. doi: 10.1007/s10703-017-0271-1. URL https://link.springer.com/article/10.1007/s10703-017-0271-1.

Srinivas Pinisetty, Partha S. Roop, Steven Smyth, Nathan Allen, Stavros Tripakis, and Reinhard Von Hanxleden. Runtime enforcement of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 16(5s), sep 2017b. ISSN 1539-9087. URL https://doi.org/10.1145/3126500.

Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, feb 2000. ISSN 1094-9224. URL https://doi.org/10.1145/353323.353382.

Sira Yongchareon, Chengfei Liu, and Xiaohui Zhao. An artifact-centric view-based approach to modeling inter-organizational business processes. pages 273–281, 01 2011. ISBN 978-3-642-24433-9. URL https://link.springer.com/chapter/10.1007/978-3-642-24434-6_22.

Xiangpeng Zhao, Jianwen Su, Hongli Yang, and Zongyan Qiu. Enforcing constraints on life cycles of business artifacts. In *2009 Third IEEE International Symposium on TASE*, pages

111–118, July 2009. doi: 10.1109/TASE.2009.46. URL https://ieeexplore.ieee.org/document/5198493.

About the authors

Gaurav Gupta received B.Tech. + M.Tech. degree in computer science and engineering from Indian Institute of Technology Bhubaneswar, India. You can contact the author at gg13@iitbbs.ac.in.

Saumya Shankar is a research scholar at School of Electrical Sciences, Indian Institute of Technology Bhubaneswar, India. You can contact the author at ss117@iitbbs.ac.in or visit https://www.iitbbs.ac.in/school-people-scholars.php?code=es.

Srinivas Pinisetty is an assistant professor at School of Electrical Sciences, Indian Institute of Technology Bhubaneswar, India. You can contact the author at spinisetty@iitbbs.ac.in or visit https://www.iitbbs.ac.in/profile.php/srinivaspinisetty/.