Mapping Study on Constraint Consistency Checking in Distributed Enterprise Systems

Tomas Cerny
tomas_cerny@baylor.edu
Computer Science, Baylor University
Waco, Texas, USA

Andrew Walker
Andrew_Walker2@baylor.edu
Computer Science, Baylor University
Waco, Texas, USA

Jan Svacina
Jan_Svacina2@baylor.edu
Computer Science, Baylor University
Waco, Texas, USA

Vincent Bushong
Vincent_Bushong1@baylor.edu
Computer Science, Baylor University
Waco, Texas, USA

Dipta Das
dipta_das1@baylor.edu
Computer Science, Baylor University
Waco, Texas, USA

Karel Frajtak
frajtak@fel.cvut.cz
CS, FEE, Czech Technical University
Prague, Czech Republic

Miroslav Bures
miroslav.bures@fel.cvut.cz
CS, FEE, Czech Technical University
Prague, Czech Republic

Pavel Tisnovsky
ptisnovs@redhat.com
Red Hat
Brno, Czech Republic

ABSTRACT

Constraint consistency errors in distributed systems can lead to fatal consequences when left unobserved and undetected. The primary goal of quality engineers should be to avoid system inconsistencies in general. However, it is typically a much more straight forward process in monolith-like systems with one codebase than in distributed solutions where heterogeneity occurs across modules. In this paper, we raise the research question of what is the existing state-of-the-art and research literature practice when it comes to consistency checking in distributed systems. We conducted a systematic search for existing work and assess the evidence to categorize the approaches and to identify used techniques. Identified works offer interesting directions and achievements. Often the works share tool prototypes and instruments to build on the top of when performing further research in this direction and we share them in this paper. Finally, we discuss open challenges and gaps in this field to promote the interest of the research audience.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**; *Parsers*; **Formal methods**; **Consistency**; • **Social and professional topics → System management**; • **Security and privacy** → *Logic and verification*; *Access control*; • **Information systems** → *Data extraction and integration.*

KEYWORDS

Constraint Consistency, Consistency Checking Distributed Systems, Mapping Study, Quality Assurance, Security Policies

RACS '20, October 13–16, 2020, Gwangju, Republic of Korea

ACM ISBN 978-1-4503-8025-6/20/10…$15.00
https://doi.org/10.1145/3400286.3418257

ACM Reference Format:
Tomas Cerny, Andrew Walker, Jan Svacina, Vincent Bushong, Dipta Das, Karel Frajtak, Miroslav Bures, and Pavel Tisnovsky. 2020. Mapping Study on Constraint Consistency Checking in Distributed Enterprise Systems. In *International Conference on Research in Adaptive and Convergent Systems (RACS '20), October 13–16, 2020, Gwangju, Republic of Korea.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3400286.3418257

1 INTRODUCTION

During software development, we often define certain system constraints that we expect the system to enforce. These constraints could be domain rules, user input validation, security policies, data restrictions, etc. We expect the defined constraints are valid in the system. However, mainstream application design does not always make it possible to define constraints separately from the program logic, which would enable efficient reuse of constraints, and instead binds constraints with particular program logic. As a result of this binding, the definition of constraints might lead to explicit restatements of the constraints across the program, leaving space for errors. For instance, when we define a constraint for an input field to restrict the maximal length of the user input value, we expect the persistence to match the setting. However, it is not necessarily true unless both the user interface and backend match in their constraint setting. Moreover, at any time the same type of input can be accepted by the system by two distinct paths. If constraint enforcement in one area is left unaddressed, it opens a space for constraint consistency errors.

Let us depict a few examples of consistency errors in a competition management system with team registration. This system might be accepting teams for a competition under a business rule restricting the team to have at most three members. This rule might be defined and enforced in the code when we create and register a new team. Still, when we edit the team, the rule code might be omitted, leaving space for registered teams with more members. Next, the business rules may state that only a coach can edit a team. However, the member removal functionality might be exposed to members, which leads to inconsistent security policy on the system methods.

Furthermore, the frontend might have an interactive interface to add teams or an Excel import feature. Initially, the system enabled the import of any sort of team name. A team manager might define an active JavaScript alert function as the team name. This incident would trigger an actual call by the system when the contest manager went to see team details causing concerns. Thus developers enforced input validation for the interactive part of the system disallowing all HTML tags. However, they forgot to address the import feature leaving the import feature unchecked. Later, a new request came to reduce the size of the team name character length. Primarily, this was restricted in the persistence level, so the interactive interface must be updated to match the persistence setting. Since there are multiple endpoints in the system, one path was mistakenly left unchanged. Then the system would allow users to import teams even when the imported names were longer than the rules specified, but it silently cropped the name lengths. These kinds of data-loss errors are hard to observe in real-time by users leaving space for their surprise upon arrival to the competition (e.g., team name "Awesome Assembly" reduced to eleven characters).

When we aim to observe the above errors, we can assess and debug the code for a specific patch only, which is rather complicated or we could perform code analysis [7, 27, 29, 30] and transform the code representation to graphs, e.g, Abstract Syntax Trees (AST), Control-Flow Graphs (CFG) [18, 28, 37], or Program Dependency Graphs (PDG) [29, 33], to analyse the consistency across graph paths [41].

However, the complexity of modern software systems continues to grow. Nowadays, the most common need is virtually infinite scalability, which is provided by cloud computing. This, however, leads into the fragmentation of the system into smaller self-contained and self-scalable modules. Microservice Architecture [6, 12] is the mainstream direction for cloud-friendly systems. These enable each microservice module to follow different development practices or use a different platform. As the individual modules interact, and assist with distributed processes, they need to preserve some level of consistency, which becomes a nightmare for quality engineers seeking consistency assurance. Naturally, code assessment is a solution but most likely very inefficient, especially since the nature of different microservice might not be homogeneous, and they might not have the knowledge of all the involved frameworks.

Perhaps code analysis is the right direction; however, to get the answers of what is the current best practice to detect constraint inconsistencies in enterprise distributed systems, we perform this systematic research mapping study [24]. We seek to find techniques and tools previously used to avoid consistency errors in these systems and aim to identify gaps in this important topic to report them to the research community as open challenges worth addressing.

The rest of this paper is organized as follows. In Section 2, we define the notions used in this paper. Section 3 walks through the settings and resulting statistics of the systematic mapping study. The actual results we analyzed are shared in Section 4. The paper closes with a conclusion that summarizes the contributions along with future work.

2 BACKGROUND

In this study, we focus on enterprise systems. Enterprise [12, 13] systems are typically seen as large and complex. These systems enable interaction with many users concurrently while enforcing business constraints and processes. This kind of software is used by a business to help the organization solve enterprise problems or to bring automation. Enterprise systems could be found in many disciplines, including healthcare, transportation, telecommunication, banking, e-commerce, power grids, and defense systems, among others. Modern enterprise solutions follow a set of development standards. Besides, their design direction is given by development frameworks that expedite the development process and simplify developer training. Enterprise solutions share similar architecture [12, 13], which evolved into a cloud-friendly industry-standard microservice architecture [6]. One can expect modern enterprise solutions to be divided into separate self-contained modules. The advantage of this approach is that each module can auto-scale independent of other modules, which makes the solutions flexible to end-user needs and responsiveness. It is also important to highlight that modern enterprise solutions fit under distributed software.

In the above paragraph, we mentioned that enterprise systems enforce certain constraints. These could be constraints placed on the data scheme, also known as integrity rules. Next, we could consider user input validation constraints, enforcing data format, e.g., email or credit card number format. Additionally, there are domain or business rules that constrain the situations and state how business objects can operate under a given system. There is a specific category that could be seen as a part of business rules or on its own, which is the security policies or access rules.

The constraints of a system can be seen as a specific set of limitations or restrictions. It can be understood as a specific state, which corresponds to an object state in object-oriented programming [14]. However, it could also mean a condition or even representation of behavior. In programming, we often speak about rules enforcing constraints. Similarly, what is not constrained is allowed.

In the Introduction section, we mentioned inconsistency and provided a few examples. Inconsistency could be understood as not being compatible with one another, which could relate to facts or claims, or commonly in software development related to conditions, policies, or rules. Inconsistency could happen because of the inclusion of additional elements making something incoherent or illogical, especially in the aspect of business rules [13]. This might not happen intentionally; it could be a result of a typological error in constructs missing language type-safety, or duplication of the constraint in multiple locations in the system (e.g., distributed modules). Software evolution could introduce such inconsistencies, especially when selected modules or subsystems are managed by distinct teams, which is the case for microservices. We could also consider what it means for a software system to be consistent. Consistency is a set of cohesive facts or claims, a harmony between states. Naturally, in software development, we target consistency within the set of constraints and aim to avoid inconsistencies.

Table 1: Search Query Results for Various Index Sites

Indexer	Found results	Used results
ACM DL	111	8
IEEE Xplore	136	5
SpringerLink	84[1]	1
ScienceDirect	105[2]	1
Total	436	15

3 MAPPING STUDY METHOD

The previous Section narrowed the target scope of this study. To find existing techniques, instruments and insight for constraint consistency in distributed enterprise systems, we consider the software engineering practice of systematic mapping studies [24].

The phases of the mapping study are as follows. We define the research questions of this study. Next, we identify terms for the search query to identify existing related works. We filter the papers that are out of scope for this study after we perform the search at various indexing sites. Finally, we assess the identified papers.
Our *research questions are as follows*:

RQ1 What is the taxonomy of current research directions?
RQ2 What techniques have been previously used or identified to check constraint consistency?
RQ3 What tools have been used to avoid consistency errors?
RQ4 What are present time open challenges and gaps?

Next, we identify the *search terms*. The research direction of constraint consistency in distributed enterprise systems drives our initial search query string for the indexing servers of ACM DL, IEEE Xplore, Spring, and ScienceDirect. However, upon experimentation and the quality assurance on the known set of literature, we extend the search terms to use possible synonyms.

We divide the search query into four parts for discussion. We search for constraints, but we consider that literature can use alternative naming, e.g., business rules, security policy, or even property. From this, we derive the search clause to be a *constraint OR rule OR property OR policy OR security* . Next, we want to filter the results to match *consistency OR inconsistency*. The nature of the related work should be in consistency verification with similar terms leading to *checking OR verification OR enforcement OR detection OR avoidance* Finally, we further filter the results to those that match enterprise distributed systems but also extended it with similar terms applied to specific architecture leading to *distributed OR cloud OR microservice*. The *full Search Query* is showed by Listing 1.

We apply the search query to *perform search* over the indexing sites. The results represent the primary set of works we assess in this study. The search results are shown in Table 1. The first column specifies the indexing site, and the second column provides the results found by the Search Query.

Next, we *filter* the primary set of works. Specifically, we filter works that are related to our intended scope and focus on distributed systems. Also, we assess the quality of the paper for selection. Finally, we exclude papers based on the title, abstract, full–text reading, and quality assessments.

We do not consider short papers with less than four pages, non-English language, papers without available full-text, non-peer-reviewed papers, books, papers without a particular contribution, e.g., opinion papers. Also, we do not consider existing work on distributed databases, nor do we consider resource consistency on a cloud platform. Also, we do not look into law compliance or compliance with provider policies. The filtered results are in Table 1 presented as the last column, which is the number of papers that made it through the filtering phase. Finally, we assess the related work sections, cited and citing papers, and identify other relevant papers that fit this study.

For all the papers that pass trough our filtering process, we perform the *mapping phase* and the content analysis. Here we extract the relevant information answering our defined research questions from the primarily selected papers. The extracted information from the selected papers is classified. Also, we analyze threats to the validity and limitations of the study. These results are shared in the next Section.

4 ANALYSIS OF RESULTS

Out of 436 papers returned by the search, we identified a small number of relevant works. Only 15 papers were selected and considered for the analysis. We list the selected papers in Table 2. The majority of the works relate to compliance with law-based policies, infrastructure, distributed data synchronization, or low-level protocols. Even out of the identified papers, we included some that are only a partial match. The major area of works we identified is security-related. However, we also identified works related to business processes, rule matching, development life-cycle, testing, and persistence. These areas correspond to the RQ1, and below, we provide more details.

4.1 Security

In the area of security, existing works look into construction of a global policy [17], suspicious operation detection [11, 23, 42], effective policy matching [19], and distinct policies applied [34, 35] across the same API implementations.

Role-Based Access Control (RBAC) is commonly used by enterprise systems to address security. Authorization resolvers consider the concept of role rather than individual users. For instance, Java EE has a security standard for this [16, 25] that provides an annotation profile for endpoints. The composition of local access control policies into a global coherent security policy based on Role-Based Access Control (RBAC) is tackled in [17]. Violations in policies may

[1]Due to the specifics of SpringerLink we have replaced AND operators with $NEAR \setminus 2$ ($NEAR \setminus 10$ for the last AND)
[2]Since ScienceDirect limits the number of boolean connectors we combined the results using BibDesk tool [21])

Listing 1: The Search Query for Research Indexing Sites

```
(constraint OR rule OR property OR policy OR security) AND (consistency OR inconsistency OR compliance)
                AND (checking OR verification OR enforcement OR detection OR avoidance)
                AND (distributed OR cloud OR microservice)
```

permit unwanted access; therefore, resolving conflicts is one of the most important issues in inter-operation policy design. Such violations can be introduced by component evolution. The work illustrates several kinds of RBAC violations that may occur in distributed systems/domains during composition (Role Inheritance Violation, Separation of Duty, Cardinality Violation, Resource sharing).

- Role Inheritance Violation - happens in a situation when the hierarchy of crossing domains opens a path that enables a particular role without inheritance relation with another role from a local domain to assume the permission of the local domain role.
- Cardinality Violation - may happen for roles, users, and shared resources with multiplicity restrictions. An example is when roles can be assigned at most n users who can access the system simultaneously.
- Separation of Duty - happens when the same user cannot be assigned conflicting roles at once. Alternatively, in reverse, the same role cannot be assigned to conflicting users.
- Resource sharing - happens when a local resource is shared with other domains under a cardinality restriction, which may lead to circular waiting and deadlock

To address these situations, the authors have adopted colored Petri nets. Each of the violations is then transformed into a graph-based problem. Moreover, they show how each can be detected by common algorithms such as shortest-path for role inheritance violation, depth-first search to detect cycles, max-flow for user role assignments. They also define the algorithm to prevent deadlocks. In a case study demonstration, they demonstrated on two-domain environments. There are no limitations to multiple-dimensional domains.

To broaden the details in the RBAC area, we have previously addressed violations in non-distributed environments [41] across overlapping executions paths if distinct endpoints using the Java standard [16]. The conflicts we identified with roles were:

- An unrelated access violation - is the most primitive and occurs when two non-linked roles access the same parts of a system.
- A hierarchy violation - is basic and occurs when two inheritance related roles access the same part of the system.
- An entity access violation - which reports the situation when access to a specific persistence entity occurs through distinct endpoints with different roles.
- An unknown role violation - occurs when an endpoint applies for a role not found on the role tree.
- An exposed public endpoint - happens when there is no security definition defined on an endpoint, which is often caused by human error.

In our case, we used code analysis to identify endpoints, and access enforcements, which can scale towards the results presented [17] but applicable to the distributed domain.

What one may notice, especially in the microservice model is that each module defines its own rules. Whether these rules correspond to security policy or business rules, the challenge is to match them when searching for conflicts. This challenge is well addressed by one of the identified works [19], where authors look into security policy matching. They present a formal definition of policies and relationships among rules. In addition, they identify various categories of matching expressed as Sets. For instance, for two rules *R1(attribute1, attribute2, attribute3, attribute4, action), R2(attribute1, attribute2, attribute3, attribute4, action)* they recognize:

- irrelevant match : $R1 \cap R2 = \emptyset$
- exact match : $R1 = R2$
- partial match :
  $\exists i (1 \leq i \leq 4), R1.attribute_i \cap R2.attribute_i \neq \emptyset$
- really contain match :
  $\forall i (1 \leq i \leq 4), R1.attribute_i \supseteq R2.attribute_i$
- associated match :
  $\forall i (1 \leq i \leq 4), R1.attribute_i \cap R2.attribute_i \neq \emptyset$ AND
  $\exists j (1 \leq j \leq 4), R1.attribute_j \not\supseteq R2.attribute_j$
  $\wedge R1.attribute_j \not\subseteq R2.attribute_j$

Furthermore, they provide conflict categorization. For instance, they recognize redundant rules with the same action and some or additional attributes. Next, they consider generalization conflict where one rule contains another, and the attribute values of the action parts are different. Moreover, they see a reverse redundancy

Table 2: Papers analysed in this study

Reference	Paper Title	Year
[17]	Secure interoperation design in multi-domains environments based on colored Petri nets	2013
[42]	Digging Evidence for Violation of Cloud Security Compliance with Knowledge Learned from Logs	2019
[19]	A Method of Conflict Detection for Security Policy Based on B+ Tree	2019
[35]	A Security Policy Oracle: Detecting Security Holes Using Multiple API Implementations	2011
[23]	DeaPS: Deep Learning-Based User-Level Proactive Security Auditing for Clouds	2019
[11]	DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning	2017
[40]	Towards Property-Based Consistency Verification	2016
[38]	Conformance checking of electronic business processes to secure distributed transactions	2013
[10]	Online and offline conformance checking of inter-organizational business processes with incomplete process log	2016
[43]	A Constraint Mechanism for Dynamic Evolution of Service Oriented Systems	2012
[3]	Test-Based Security Certification of Composite Services	2018
[20]	A Mobile Agent Approach for Global Database Constraint Checking	2004
[22]	Flexible Consistency Checking	2003
[9]	Efficient Detection of Inconsistencies in a Multi-Developer Engineering Environment	2016
[39]	Live and Global Consistency Checking in a Collaborative Engineering Environment	2019

where one role contains another, and the attribute values of the action part are the same. Apart from this, there is also coverage redundancy wherein one rule contains another, and the attribute values of the action part are different. Next, they define a partial redundancy with two rules being associated matches, and the attribute values of the action part are the same. Furthermore, they recognize an associated conflict with two rules being associated matches, and the attribute values of the action part are different. Finally, there is no conflict for an irrelevant match. A great contribution of the paper is the proposed B+ tree-based policy conflict detection algorithm. Since we can expect a large number of rules in distributed management systems, detecting policy conflicts brings performance issues. The proposed method constructs the rule attribute index based on an efficient indexing structure (B+ tree). All the recognized policy library rules are inserted into an index tree, uniquely mapping each rule, and speeding up the process of conflict detection. In the presented evaluation, a B+ tree-based policy conflict detection time maintained at about 0.3 milliseconds, no matter the number of rules in the system.

Quite a different evaluation perspective is runtime detection. The next considered work [42] uses an interesting perspective involving log analysis, which we believe could be used in a dynamic environment. The goal is to detect suspicious cloud user operations that cause security compliance violations. In this work, system logs are grouped, extracted, and parsed into log event sequences. These then represent cloud behaviors related to user operations. The execution traces are recovered and labeled from the log. These traces are transformed to vectors for feature representation, in particular, authors suggest using N-gram [32] and Term Frequency - Inverse Document Frequency [31] techniques. With this, classifiers can be trained with labeled vectors from normal system behavior for automatic recognition of suspicious operation requests. When traces are recognized as suspicious operation requests, the normal, abnormal, and predicted log events are marked in these traces as evidence of compliance violations for auditors. In a case study, the authors present several security compliance scenarios. Even though the work does not share a prototype, it provides a comparison with other log analysis techniques that could be used for log analysis. Among those mentions DeepLog [11] uses deep learning, Long Short-Term Memory (LSTM) neural network. They train the model on a normal system executions log and indicate anomaly on a trained model. DeepLog builds on top of Keras open-source neural network library for Python and shares the project. Another similar work DeaPS [23] was identified in our search. It uses an identical LSTM approach to DeepLog as proactive security auditing in the cloud.

The final security perspective looks into an unusual use case. It considers distinct implementations of the same API to detect security policy compliance errors. In [35], authors suggest that derivation of the policy by code-mining utilizing patters might be insufficient. They suggest that the analytical system should be dynamic. The work extracts the security policies dynamically from at least two different implementations of the same API to compare their access to system operations and other resources and identify differences. They utilized flow- and context-sensitive analysis. In particular, they take as an input the API definition and multiple of its implementations along with the definitions of security checks and security-sensitive events. The starting point of the process is each API endpoint that they analyze in-depth for the applied checks to derive the policy and compare the results across distinct implementations. In a study, they found 31 severe issues in various Java libraries, e.g., those used in enterprise systems following a particular specification. While this work lacks the distributed system aspect, it brings an interesting perspective. This use case is not transferable to systems with a single implementation; however, it could be a foundational concept for further research as it provides in-depth detail to construct security policy oracles. An extension of this work [34] has been applied to access control policies in enterprise applications. It considers static analysis to compute what they call an inter- procedural access-control template. Such a template includes all program statements involved in this instance of access-control logic. Next, it finds faulty access-control logic that misses some or all of these statements and inserts the missing statements to the given code.

4.2 Other Research Areas

4.2.1 Property-based testing. One of the research perspectives on test consistency uses property-based testing [40]. This declarative approach enables static checking of correctness conditions. The work adopts a first-order logic predicates for graph entity representation. What needs to be provided as input are application-level properties that should be satisfied along with the generic format of valid input. Proposed property-based consistency verification framework Conver then generates test cases with random inputs to the program under test while collecting details of all involved operations and verifying the validity of the supplied properties throughout the execution. Next, it derives a graph of entities describing the client-side outcomes, global ordering, and visibility of events. Based on [40], the framework verifies the compliance to a given consistency model by building and validating the required entities for all logic terms composing the entire consistency predicate. However, a significant limitation of this approach is that it requires manually written property-like specifications of the system.

4.2.2 Distributed Business Processes. Distributed Business Processes are occasionally found in this research direction. In [38], authors consider conformance checking as an area of process mining. They aim to verify that the execution of a distributed business process satisfies specifications represented by formal models. A software agent intercepts the communication of each module for this purpose. Agents send the messages to a centralized validation authority to collect information from particular modules about given processes. This could be seen as similar to using distributed logging. Here, the validation authority merges the events to a process tree, which is then extended with validation rules to a validation tree that represented possible paths of the particular process. The verification uses the validation tree, which is traversed, and the validation rules contained in the node are verified. An extended work [10] of the same authors considers incomplete event logs. They suggest utilizing it for security verification and for anomaly detection, which leads to the log-based anomaly detection discussed in the security subsection [11, 23, 42].

Business processes have been mentioned more often in the context of our search. For instance, in [43], service-oriented architecture is considered for topological constraints and their satisfaction. The work implements a verification tool to model runtime application constraints and verify consistency to facilitate evolution. However, the goal is to evaluate behavior and topology. They use Graph Grammars with Neighborhood Controlled Embedding for which they consider Web Service Business Process Execution Language (WS-BPEL)-based process description and involve Web Service Definition Language(WSDL) documents. Similarly, in [3], the authors considered the verification of security properties of composite services. They considered extensions of test-based security certification for composition. The target domain of the approach is in the "Business Process as a Service" cloud model. The solution idea is to derive a virtual test-based security certificate for a service composition based on the certificates of its component services. The composite service evidence proving the property is inferred from the evidence initially used to certify the individual components. The technique then uses derived of machine-readable virtual certificates along with virtual test cases and their quantitative evaluation of the quality of the virtual certification process and corresponding service composition. Even here, service specifications are used (e.g., WSDL), along with BPEL. This idea of virtual qualities built from individual services qualities seems promising for possible future research directions.

4.2.3 Constraints in Multi-bases. Constraint checking of multibase with multiple instances of a database on distinct mobile devices is considered in [20]. This work discusses another possible direction a meta-model for constraint exchange. For databases, they consider the enforcements related to insert, update, and delete operations. This could be the same in the case of a software system. A multibase with multiple instances of a database on distinct mobile devices interacts and integrates with a federated database. This scenario has a great likelihood of a global constraint violation. To address this use case, mobile agents periodically retrieve global meta-data from the federating database to perform local constraint checking. A similar approach can be applied for constraint verification in an enterprise where it is important to collect information from distributed peers; to form a virtual overlay perspective.

4.2.4 Artifact Consistency. Another research perspective considers consistency across artifacts and development process phases. In [22], authors considered checking the consistency of heterogeneous distributed documents and demonstrate the application on Java Enterprise Edition projects. This work carries out some requirements and observations usable in nowadays distributed solutions. The paper's focus is to evaluate consistency across the design, implementation, and deployment of a given system along with incremental checking upon evolution. The authors propose a new xlinkit framework for this task. They derive a set of requirements for consistency management services. These consist of "flexibility in constraint application and a tolerant approach to consistency; support for distributed documents; a mechanism for bridging the heterogeneity gaps between different specification languages, without resorting to a common vocabulary; and strong diagnostics that show which parts of specifications contribute to inconsistency." The approach utilizes XML-based technologies turning the documents into a Document Object Model (DOM) (a tree structure). The work use legacy XML application deployment descriptors in addition to file descriptors. Both these correspond to the current technology approach, YAML deployment descriptor files, or annotation descriptors. Each found element is linked to the source or to a file where it reappears. They recognize horizontal and vertical constraints. In particular, they define them as follows:

- "horizontal constraints between artifacts at the same stage of the process, and
- vertical constraints between stages, for example, between the design and implementation".

Authors observe that constraints can be expressed in UML through stereotypes and tagged values. Moreover, they recognize four types of constraints:

- standard - given by the specification language,
- extension - where additional semantics is needed,
- integration - for cross-platform interaction,
- and custom - for organization-specific conventions.

Moreover, they observe that specification languages such as Object Constraint Language (OCL) are rather limited when it comes to diagnostics feedback or issues. Since OCL returns true or false, it makes it hard to locate the cause of the issue. Their proposed tool xlinkit uses a simple language based on first-order logic involving XPath expressions. Thus it recognizes XML structures and uses static analysis for Java checking along the tree paths recognizing intersections.

A similar approach to xlinkit looks into the global consistency of distributed artifacts ported into a cloud called DesignSpace. It is detailed in [9, 39]. This direction moves towards Java code and UML model artifact consistency. It transforms the artifacts into a uniform representation. For the XML, it deals with parsing, and for Java, it used Reflection API[4], also referenced as metaprogramming. Besides, linking across artifacts is achieved by manually populated trace matrices. For rule definitions, the work considers an OCL-like specification with context (starting point of a reference) and conditions to be met (e.g., an attribute must exist in all linked attributes). Consistency rules are defined for checking and performed upon changes in the environment.

4.3 Discussion

From analyzed works, we can answer RQ1 in different research directions. The major direction is security, in particular policy evaluation [17, 19, 35] along with anomaly detection [11, 23, 42]. Security direction is followed by distributed processes checking [3, 10, 38, 43], distributed artifact verification [9, 22, 39], rule matching [19], testing towards specification [40] and persistence [20]. Clearly, the current space for consistency checking in microservices is wide-open. From what we have discovered, it is more common to verify the system towards a given specification than to compare constraint replicas in the system. As suggested by [35] it might be a better direction to derive information rather than to pay attention to pattern-based matching or to verify towards the specification.

Regards previously used techniques and RQ2 we found a broad range of directions, including checking with XML-based specifications [9, 22], extracting an oracle [35] from the system via code analysis or deriving a virtual composite model [3]. Some approaches

Table 3: Existing tools

Tool	Purpose	Web link	Reference
Soot	Soot static analysis framework	http://www.sable.mcgill.ca/soot/	[35]
Xlinkit	Software artifact sychronization	http://xml.coverpages.org/ni2001-02-27-d.html	[22]
DesignSpace	Software artifact sychronization	isse.jku.at/tools/dsspc/xadr.zip (pw: dsisse)	[9, 39]
Conver	Property-based consistency verification framework	https://github. com/pviotti/conver	[40]
N-gram	R package for constructing n-grams	https://github.com/wrathematics/ngram	[42]
Keras	Deep learning API	https://keras.io/	[42]
DeepLog	DeepLog's log key anomaly detection mode	https://github.com/wuyifan18/DeepLog	[11]

even suggest using the meta-model perspective [20, 39]. This meta-model perspective could be well-combined with the virtual global model. Among others we noticed log analysis [11, 23, 42] and event interception [38], which is just another form of logging. Code analysis [35] can be very efficient with access to system module details. Alternatively, research considered markup, typically XML-based, specification processing [3, 10, 22, 38, 39, 43]. We observed interesting perspectives using N-grams, Term Frequency - Inverse Document Frequency, LSTM neural networks [11, 23, 42]. We noticed various data structures involved, such as transformation to graphs, tree structures, or B+ trees, colored Petri nets [17]. All mentioned directions might help with further research since we recognized parts which have been addressed along with existing solutions.

Next, we identified various tools to answer RQ3. We noted the mentioned tools and listed them in Table 3. We also associated them with a particular research paper that mentions them.

Perhaps most challenging to answer is RQ4 related to current open challenges and gaps. The identified volume of relevant works is rather small. However, we believe in the importance of this research perspective, especially since the growing adoption of microservice architecture leads to self-deployable modules that, to some extent, interact and redefine certain constraints. Since modules are managed by distinct teams who coordinate their work over the API, constraint consistency issues might be especially evident upon module evolution. Consistency conflict resolution should be addressed in a better way than manually, which is error-prone and tedious. Log analysis and event tracing seem to be a direction that can identify conflicts that happen in runtime, and with proper mechanisms in place, it could mitigate error impacts. However, the current approaches consider deep learning, and clearly, more options exist. Distributed system modules have their log that can be easily centralized (e.g., via Kafka [15], and thus this approach could be easily adopted. However, to prevent issues in runtime, a static analysis could be considered, which leads to code analysis. However, the current work in a distributed environment is marginal. Perhaps local analysis could be performed as part of building a virtual global system overlay, as suggested by composite service certification in [3]. This opens a significant potential for research, and if accomplished, we could transform many non-distributed approaches to the distributed environment. However, this direction needs to cope with heterogeneity since modules may use distinct frameworks, languages, or design practices, and this is where we could take advantage of experience from heterogeneous and distributes artifact consistency tracking [9, 22, 39], although the link population should be automated. A new development model for distributed systems could also be a research direction. Nevertheless, in our opinion, this would hardly be adopted by industry; it is better to comply with, and extend, the existing practice. Besides, we must assume that different modules may apply different rules, and rule extraction from code could be addressed. We noticed that graph representation might be used [17] and that there exists an efficient mechanism to look-up and match rules [19]. However, it must be noted that constraints and variables might differ across modules. For instance, the same-named access role in two distinct services might mean two distinct things since separate definitions and resolvers of these roles exist in the modules. A simple name matching is insufficient in a distributed environment. Automation in distributed access role merging should be researched.

4.3.1 Threats to Validity. One threat in survey studies is inadequate coverage. This study searched through four major research indexing sites recognized by literature [24]. We followed a practice well accepted for performing mapping studies. However, we did not include statistics on authors, countries, conferences, etc. Since we performed manual selection and filtering, it might be possible that related work has been skipped. Still, we intended to mitigate it by assessment of work references and citation analysis. Similarly, data extraction could be threatened by the human factor; this was addressed by multi-peer evaluation.

5 CONCLUSION

This study looked into existing research on constraint consistency error detection in distributed systems. It provides an analysis of exiting research directions, techniques, and instruments that one could build on top of to contribute to this field. The addressed perspective is aimed to motivate research peers in conducting new research in these directions. To support research peers, we discussed open challenges that are currently under-addressed by research or completely missing but needed for efficient enterprise system evaluation. This work identifies some critical achievements that others could build on top of rather than reinventing the wheel.

Our long term overarching research goal is to develop a new automated approach for software architecture reconstruction of distributed systems [2, 26]. Our ongoing work is capable of basic multi-module integration based on data models or based on remote calls. This survey helped us identify existing work in the area of constraint consistency detection, and our virtual overlay could be applied to it. Besides we performed other similar studies towards our long term goals [5, 8, 36]. In a short term perspective, we will extend our previous work on security policy evaluation [41] and assess fit for the internet of things domain [1].

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1854049 and a grant from Red Hat Research https://research.redhat.com.

REFERENCES

[1] B. S. Ahmed, M. Bures, K. Frajtak, and T. Cerny. 2019. Aspects of Quality in Internet of Things (IoT) Solutions: A Systematic Mapping Study. *IEEE Access* 7 (2019), 13758–13780.

[2] N. Alshuqayran, N. Ali, and R. Evans. 2018. Towards Micro Service Architecture Recovery: An Empirical Study. In *2018 IEEE International Conference on Software Architecture (ICSA)*. 47–4709.

[3] Marco Anisetti, Claudio Ardagna, Ernesto Damiani, and Gianluca Polegri. 2018. Test-Based Security Certification of Composite Services. *ACM Trans. Web* 13, 1, Article 3 (Dec. 2018), 43 pages. https://doi.org/10.1145/3267468

[4] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 1996. *Pattern-oriented software architecture: a system of patterns.* John Wiley & Sons, Inc., New York, NY, USA.

[5] Vincent Bushong, Russell Sanders, Jacob Curtis, Mark Du, Tomas Cerny, Karel Frajtak, Miroslav Bures, Pavel Tisnovsky, and Dongwan Shin. 2020. On Matching Log Analysis to Source Code: A Systematic Mapping Study. In *International Conference on Research in Adaptive and Convergent Systems(RACS '20) (RACS '20)*. ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/3400286.3418262

[6] Tomas Cerny, Michael J. Donahoo, and Michal Trnka. 2018. Contextual Understanding of Microservice Architecture: Current and Future Directions. *SIGAPP Appl. Comput. Rev.* 17, 4 (2018), 29–45. https://doi.org/10.1145/3183628.3183631

[7] Tomas Cerny, Jan Svacina, Dipta Das, Vincent Bushong, Miroslav Bures, Pavel Tisnovsky, Karel Frajtak, Dongwan Shin, and Jun Huang. 2020. On Code Analysis Opportunities and Challenges for Enterprise Systems and Microservices. *IEEE Access* (2020), 1–22. https://doi.org/10.1109/ACCESS.2020.3019985

[8] Dipta Das, Micah Schiewe, Elizabeth Brighton, Mark Fuller, Tomas Cerny, Miroslav Bures, Karel Frajtak, Dongwan Shin, and Pavel Tisnovsky. 2020. Failure Prediction by Utilizing Log Analysis: A Systematic Mapping Study. In *International Conference on Research in Adaptive and Convergent Systems(RACS '20) (RACS '20)*. ACM, New York, NY, USA, 1–7. https://doi.org/10.1145/3400286.3418263

[9] Andreas Demuth, Markus Riedl-Ehrenleitner, and Alexander Egyed. 2016. Efficient Detection of Inconsistencies in a Multi-Developer Engineering Environment. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE 2016)*. Association for Computing Machinery, New York, NY, USA, 590–601. https://doi.org/10.1145/2970276.2970304

[10] A. C. D'Iddio, C. H. Schunck, F. Arcieri, and M. Talamo. 2016. Online and offline conformance checking of inter-organizational business processes with incomplete process logs. In *2016 IEEE International Carnahan Conference on Security Technology (ICCST)*. 1–8.

[11] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1285–1298. https://doi.org/10.1145/3133956.3134015

[12] K. Finnigan. 2018. *Enterprise Java Microservices.* Manning Publications. https://books.google.com/books?id=KaSNswEACAAJ

[13] Martin Fowler. 2002. *Patterns of Enterprise Application Architecture.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[14] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[15] Nishant Garg. 2013. *Apache Kafka.* Packt Publishing.

[16] Will Hopkins. 2009. JSR 375: JavaTM EE Security API. (November 2009). Retrieved March 27, 2020 from https://jcp.org/en/jsr/detail?id=375

[17] Hejiao Huang and Hélène Kirchner. 2013. Secure interoperation design in multi-domains environments based on colored Petri nets. *Information Sciences* 221 (2013), 591 – 606. https://doi.org/10.1016/j.ins.2012.09.027

[18] K. S. Kumar and D. Malathi. 2017. A Novel Method to Find Time Complexity of an Algorithm by Using Control Flow Graph. In *2017 International Conference on Technical Advancements in Computers and Communications (ICTACC)*. 66–68. https://doi.org/10.1109/ICTACC.2017.26

[19] X. Luo and Y. Lu. 2019. A Method of Conflict Detection for Security Policy Based on B+ Tree. In *2019 IEEE Fourth International Conference on Data Science in Cyberspace (DSC)*. 466–472.

[20] Praveen Madiraju and Rajshekhar Sunderraman. 2004. A Mobile Agent Approach for Global Database Constraint Checking. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC '04)*. Association for Computing Machinery, New York, NY, USA, 679–683. https://doi.org/10.1145/967900.968043

[21] M McCracken, A Maxwell, and C Hofman. 2015. BibDesk. (2015).

[22] Christian Nentwich, Wolfgang Emmerich, Anthony Finkelsteiin, and Ernst Ellmer. 2003. Flexible Consistency Checking. *ACM Trans. Softw. Eng. Methodol.* 12, 1 (Jan. 2003), 28–63. https://doi.org/10.1145/839268.839271

[23] M. Ou, L. Wang, and H. Xun. 2019. DeaPS: Deep Learning-Based User-Level Proactive Security Auditing for Clouds. In *2019 IEEE Global Communications Conference (GLOBECOM)*. 1–6.

[24] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology* 64, Supplement C (2015), 1 – 18. https://doi.org/10.1016/j.infsof.2015.03.007

[25] Pivotal. 2019. Spring Security. (2019). Retrieved March 27, 2020 from https://spring.io/projects/spring-security

[26] Florian Rademacher, Sabine Sachweh, and Albert Zündorf. 2020. A Modeling Method for Systematic Architecture Reconstruction of Microservice-Based Software Systems. In *Enterprise, Business-Process and Information Systems Modeling*. Springer International Publishing, Cham, 311–326.

[27] Dhavleesh Rattan, Rajesh Bhatia, and Maninder Singh. 2013. Software clone detection: A systematic review. *Information and Software Technology* 55, 7 (2013), 1165 – 1199. https://doi.org/10.1016/j.infsof.2013.01.008

[28] José Carlos Bregieiro Ribeiro, Francisco Fernández de Vega, and Mário Zenha-Rela. 2007. Using Dynamic Analysis Of Java Bytecode For Evolutionary Object-Oriented Unit Testing. In *25th Brazilian Symposium on Computer Networks and Distributed Systems (SBRC)*. 143–156.

[29] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. 2009. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. *Sci. Comput. Program.* 74, 7 (May 2009), 470–495. https://doi.org/10.1016/j.scico.2009.02.007

[30] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K. Roy, and Cristina V. Lopes. 2016. SourcererCC: Scaling Code Clone Detection to Big-code. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16)*. ACM, New York, NY, USA, 1157–1168. https://doi.org/10.1145/2884781.2884877

[31] Claude Sammut and Geoffrey I. Webb (Eds.). 2010. *TF–IDF.* Springer US, Boston, MA, 986–987. https://doi.org/10.1007/978-0-387-30164-8_832

[32] Drew Schmidt and Christian Heckendorf. 2017. ngram: Fast n-Gram Tokenization. (2017). https://cran.r-project.org/package=ngram R package version 3.0.4.

[33] G. M. K. Selim, K. C. Foo, and Y. Zou. 2010. Enhancing Source-Based Clone Detection Using Intermediate Representation. In *2010 17th Working Conference on Reverse Engineering*. 227–236. https://doi.org/10.1109/WCRE.2010.33

[34] Sooel Son, Vitaly Shmatikov, and Kathryn S McKinley. 2013. FixMeUp: Repairing Access-Control Bugs in Web Applications. In *Network and Distributed System Security Symposium (NDSS)*.

[35] Varun Srivastava, Michael D. Bond, Kathryn S. McKinley, and Vitaly Shmatikov. 2011. A Security Policy Oracle: Detecting Security Holes Using Multiple API Implementations. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '11)*. ACM, New York, NY, USA, 343–354. https://doi.org/10.1145/1993498.1993539

[36] Jan Svacina, Jackson Raffety, Connor Woodahl, Stone Brooklynn, Tomas Cerny, Miroslav Bures, Karel Frajtak, Dongwan Shin, and Pavel Tisnovsky. 2020. On Vulnerability and Security Log analysis: A Systematic Literature Review on Recent Trends. In *International Conference on Research in Adaptive and Convergent Systems (RACS '20)*. ACM, 1–6. https://doi.org/10.1145/3400286.3418261

[37] Muhammad M. Syaikhuddin, Choirul Anam, Ade R. Rinaldi, and Moch E. B. Conoras. 2018. Conventional Software Testing Using White Box Method. *Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control* 3, 1 (2018), 65–72. https://doi.org/10.22219/kinetik.v3i1.231

[38] M. Talamo, F. Arcieri, C. H. Schunck, and A. C. D'Iddio. 2013. Conformance checking of electronic business processes to secure distributed transactions. In *2013 47th International Carnahan Conference on Security Technology (ICCST)*. 1–6.

[39] Michael Alexander Tröls, Atif Mashkoor, and Alexander Egyed. 2019. Live and Global Consistency Checking in a Collaborative Engineering Environment. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19)*. Association for Computing Machinery, New York, NY, USA, 1776–1785. https://doi.org/10.1145/3297280.3297454

[40] Paolo Viotti, Christopher Meiklejohn, and Marko Vukolić. 2016. Towards Property-Based Consistency Verification. In *Proceedings of the 2nd Workshop on the Principles and Practice of Consistency for Distributed Data (PaPoC '16)*. Association for Computing Machinery, New York, NY, USA, Article 1, 4 pages. https://doi.org/10.1145/2911151.2911162

[41] Andrew Walker, Jan Svacina, Johnathan Simmons, and Tomas Cerny. 2020. On Automated Role-Based Access Control Assessment in Enterprise Systems. In *Information Science and Applications*, Kuinam J. Kim and Hye-Young Kim (Eds.). Springer Singapore, Singapore, 375–385.

[42] Yue Yuan, Anuhan Torgonshar, Wenchang Shi, Bin Liang, and Bo Qin. 2019. Digging Evidence for Violation of Cloud Security Compliance with Knowledge Learned from Logs. In *Trusted Computing and Information Security*. Springer, 318–337.

[43] B. Zhao, Y. Zhao, and D. Ma. 2012. A Constraint Mechanism for Dynamic Evolution of Service Oriented Systems. In *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. 103–110.