frontiers in Sustainability

ORIGINAL RESEARCH
published: 15 February 2022
doi: 10.3389/frsus.2022.792103

CIRCE: Architectural Patterns for Circular and Trustworthy By-Design IoT Orchestrations

Manos Papoutsakis[1,2]*, *Konstantinos Fysarakis*[3], *Emmanouil Michalodimitrakis*[1], *George Spanoudakis*[2] and *Sotiris Ioannidis*[4]

[1] Institute of Computer Science, Foundation for Research and Technology, Heraklion, Greece, [2] Department of Computer Science, City University of London, London, United Kingdom, [3] Sphynx Analytics Limited, Nicosia, Cyprus, [4] School of Electrical and Computer Engineering, Technical University of Crete, Chania, Greece

The adoption of Internet of Things (IoT) devices, applications and services gradually transform our everyday lives. In parallel, the transition from linear to circular economic (CE) models provide an even more fertile ground for novel types of services, and the update and enrichment of legacy ones. To fully realize the potential of the interplay between IoT and CE, the design-time definition of IoT orchestrations with proven circularity properties, and the run-time management of these orchestrations based on said properties, is of paramount importance. Nevertheless, the circularity requirements and associated properties are not only difficult to achieve at the IoT orchestration design and deployment initialization phases, but also hard to prove and maintain at run-time. Motivated by this, this paper presents the CIRCE framework for circular and trustworthy by-design IoT orchestrations. The CIRCE approach leverages concepts from pattern-driven engineering, whereby patterns are used to encode proven dependencies between the Location, Condition, and Availability (LCA) properties of individual smart objects and corresponding properties of orchestrations (compositions) involving them. These are augmented by patterns encoding trustworthiness-related properties, namely Connectivity, Security, Privacy, Dependability, and Interoperability (CSPDI). Thereby, these patterns are used to generate IoT orchestrations with proven LCA and CSPDI properties, as needed, at design time. At runtime, these properties are monitored in real-time, leveraging reasoning engines deployed across system layers, triggering adaptations to return the deployed orchestration to the desired LCA and CSPDI states, when required. Details are provided on the above novel combination of IoT, CE and pattern-based engineering, along with a proposed architecture and implementation approach. Furthermore, an assessment of a proof-of-concept implementation is provided, validating the feasibility of the proposed approach.

Keywords: circular economy, circularity properties, Internet of Things (IoT), sustainable IoT services, IoT compositions, green computing, pattern engineering, security

OPEN ACCESS

Edited by:
Vasilis Katos, Bournemouth University, United Kingdom

Reviewed by:
Nathan Clarke, University of Plymouth, United Kingdom
Christos Verikoukis, University of Patras, Greece

***Correspondence:**
Manos Papoutsakis
paputsak@ics.forth.gr

Specialty section:
This article was submitted to Circular Economy, a section of the journal Frontiers in Sustainability

Received: *09 October 2021*
Accepted: *20 January 2022*
Published: *15 February 2022*

Citation:
Papoutsakis M, Fysarakis K, Michalodimitrakis E, Spanoudakis G and Ioannidis S (2022) CIRCE: Architectural Patterns for Circular and Trustworthy By-Design IoT Orchestrations. Front. Sustain. 3:792103. doi: 10.3389/frsus.2022.792103

INTRODUCTION

Climate change, i.e., the evident shift in climate patterns mainly attributed to greenhouse gas emissions from natural systems and human activities, has local and global effects, forcing all living organisms, including humans and our societies, to adapt, in order to deal with its impact (Fawzy et al., 2020). When considered in conjunction with the depletion of our planet's finite resources, exacerbated by the ever-increasing consumption and global waste generation [according to the World Bank Group, solid waste per day will reach 6.5 million tons by 2025 The World Bank, 2019], it becomes evident that the shift to newer, more sustainable models is a pressing, existential requirement for humanity. In this landscape, Circular Economy (CE) has emerged in recent years, motivating the transition of our societies from the old, linear "take-make-dispose" model to an economy that is restorative and regenerative by design, featuring a continuous "Reduce-Reuse-Recycle" (3R) cycle that aims to keep products, components and materials at their highest utility and value at all times (Ellen MacArthur Foundation the McKinsey Center for Business Environment, 2015).

IoT and CE can create a promising synergy, where the former offers knowledge about available resources (assets) regarding their availability, condition, and location; and the latter allows for more efficient use of said resources by the extension of their lifetime and the maximization of their utilization. This interplay, where IoT objects with proven key circularity properties maximize IoT resource and data harvesting, is highlighted in Askoxylakis (2018). This synergy can be applied in many concepts such as smart agriculture, smart cities, etc. For example, agribusiness moves toward precision agriculture, trying to make their complex-by-nature operations more sustainable and efficient. IoT application and CE concepts can be key enablers toward this direction. Furthermore, authors in Del Borghi et al. (2014) perform an analysis of best practices on smart waste management in the context of the circular economy initiative "LiguriaCircular." According to this analysis, ICT applications can smartly enhance the visualization of intelligent waste management systems. A number of opportunities for CE (3R principles) in energizing smart cities are presented in Musti (2020), including a Demand Side Management, waste from solar PV industry, repurposing electrical vehicles, recycling the batteries, and heavy oil recycling and heat recovery. The Smart CE framework is introduced in Kristoffersen et al. (2020) that allows for translating the circular strategies into the business analytics requirements of digital technologies.

Overall, CE is being actively investigated both from a theoretical (Lahti et al., 2018) and implementation (Kalmykova et al., 2018) perspective. Nevertheless, the potential stemming from the interplay of CE with the paramount developments taking place in parallel on the Information and Communication Technologies (ICT) front, namely the adoption of Internet of Things (IoT) products and services and 5G communication networks, each enabling new capabilities, business models and services, needs to be studied more extensively (Miaoudakis et al., 2020). In this context, an important enabler to fully exploiting the potential of this interplay would be the capability for design-time definition of IoT orchestrations with proven circularity properties (along with other properties—e.g., related to trustworthiness and reliability—if possible) and the run-time management of these orchestrations based on said properties.

Furthermore, there are some important business and technical challenges that need to be addressed to allow these technologies to reach their full potential (Lee and Lee, 2015; Botta et al., 2016; Razzaque et al., 2016), such as:

- **Dynamicity**—The dynamic nature of IoT dictates dynamically adaptive behavior at runtime at all layers (IoT infrastructure, IoT applications, IoT smart objects).
- **Scalability**—The growing number of connected users, objects and applications requires high scalability of the IoT infrastructure and network. Scalability at the infrastructure level demand discovery and orchestration of smart objects, event processing and analytics, even integration of IoT platforms. At the network level, the increased demand claim for programmable connectivity and service provisioning in a way that guarantee end-to-end optimizations, based on the desirable application requirements.
- **Heterogeneity**—Semantic interoperability is translated into three tasks: (i) definition of capabilities and constraints of heterogeneous smart objects, (ii) interpretation of the generated data, and (iii) establishment of meaningful connections between heterogeneous IoT platforms. Despite the existing standardization efforts, Semantic interoperability is still a challenge for IoT applications.
- **End-to-end Security and Privacy**—Preservation of security and privacy properties is a challenge since the large number of distinct smart objects in a complex IoT composition makes it difficult to (i) analyse all the potential vulnerabilities, (ii) select appropriate control mechanisms, and (iii) preserve desired properties due to dynamic changes in applications and security incidents.

The above highlight the importance of being able to verify (at design-time, and at runtime, if possible) properties such as the connectivity, security (i.e., confidentiality, integrity, and availability), privacy, dependability, and the interoperability of IoT orchestrations and their underlying components.

While some of these challenges have been studied from an IoT perspective, only a small cluster of research efforts (where the authors of this manuscript have been involved), has focused on exploring the above holistically, also considering the CE-IoT interplay and the associated applications and services that could be enabled. Two research projects were the first to highlight the potential of the CE and IoT interplay, albeit without considering 5G networks, and have also partly motivated CIRCE; these are CE-IoT[1] and Ideal-Cities[2] The former explicitly aims to investigate novel ways in which the CE and IoT interaction can drastically change the nature of products,

[1] Available online at: https://www.ce-iot.eu/.
[2] Available online at: https://www.ideal-cities.eu/.

services, business models and ecosystems, while the latter focuses mostly on the provision of trustworthy IoT Participatory sensing applications, but also considering circularity-aware smart city asset management. Furthermore, both projects take a pattern-driven approach to the definition of circularity and other relevant properties: CE-IoT focuses on LCA and CSPDI properties (CE-IoT, 2020), as the work presented herein, while Ideal-Cities defines CRSP Patterns (Circularity, Resilience, Security, Privacy) (Ideal-Cities, 2020). While these research efforts, and especially CE-IoT, motivated and lay the foundations for CIRCE, they only considered smart object compositions that satisfy said properties, but not the design and runtime specification of higher-level orchestrations to support specific IoT applications. Further, they did not inherently consider the co-existence and potential of 5G networks, nor did they feature cross-layer reasoning capabilities, separating edge, network and backend assets and properties. Furthermore, they have not, at the time of writing, defined patterns for all properties and sub-properties, while early implementations were limited in scope, not spanning cross-layer interactions, and relying on a different approach for property reasoning, based on Event Calculus (Hatzivasilis et al., 2019).

Motivated by the above, this manuscript presents "**CIRCE,**" a framework leveraging ar**ChI**tectural patte**R**ns for **C**ircular & Trustworthy by-design IoT orch**E**strations. CIRCE leverages concepts from pattern-driven engineering, whereby patterns are used to encode proven dependencies between specific circularity properties (namely: **Location, Condition, and Availability**, referred to as **LCA**) along with trustworthiness-related properties (namely: **Connectivity, Security, Privacy, Dependability, and Interoperability**, referred to as **CSPDI**) of individual smart objects and corresponding properties of orchestrations (compositions) involving them. These patterns are then used:

- **at design-time**, to generate IoT orchestrations that provably satisfy the required LCA and CSPDI properties
- **at run-time**, monitoring these properties in real-time, through reasoning engines deployed across IoT layers (edge, network, and backend), triggering adaptations to return the deployed orchestration to the desired LCA and CSPDI states, when required.

Along with a proposed architecture and implementation approach. Furthermore, an assessment of a proof-of-concept implementation is provided, validating the feasibility of the proposed approach.

To present and validate the CIRCE approach, this paper is organized as follows: section Materials and Methods presents the materials (such as the background on circularity properties and pattern-driven IoT orchestrations) and methods (including the pattern language defined for supporting CIRCE, and the automated pattern reasoning approach adopted); Section Results presents the results, including the high level CIRCE architecture, the exemplary set of defined CSPDI & LCA patterns covering all properties, and the evaluation results of the Proof-of-Concept (PoC); finally, Section 4 includes the discussion and concluding remarks.

MATERIALS AND METHODS

Materials

Architectural Patterns

Patterns are re-usable solutions to common problems and building blocks to architectures (Schumacher, 2003), the foundations of which were laid by the architect Christopher Alexander in his seminal work "The Timeless Way of Building" (Alexander, 1979). Patterns have gained significant attention by the research community for quite a few years now and the result of this attention is a plethora of patterns delivered in different forms such as books, catalogs, and the academic literature.

A recent survey by the authors (Papoutsakis et al., 2021a) aggregates a number of such patterns focusing on security and privacy properties. Some notable works on trust-related aspects defined through patterns are also provided herein. Authors in Steel et al. (2005) introduce a catalog of 23 security patterns, focusing only on Java 2 Platform Enterprise Edition (J2EE) applications, Web services, and identity management. They adopt a developer-centric approach to patterns' specification. Forty-six additional security patterns (Schumacher et al., 2006) in the form of a book cover areas such as enterprise security and risk management, identification and authentication, access control, accounting, firewall architecture, and secure internet applications. One of the most recent literature studies regarding privacy patterns research (Lenhard et al., 2017) presented 148 privacy patterns. Additional work, mentioned below, adds to that set of available privacy patterns. Authors in Chung et al. (2004) created a pattern language and expressed 45 pre-patterns describing application genres, physical-virtual spaces, interaction, and system techniques for managing privacy and techniques for fluid interactions. The design of Privacy Enhancing Technologies (PETs) utilizing corresponding patterns was the aim of Hafiz (2013), contributing 12 patterns. The privacy pattern catalog presented in Drozd (2015) uses a classification based on the description of the privacy principles within the international standard ISO/IEC 29100:2011 (2011).

Composition of IoT Devices and Services

There is a large body of works regarding the composition of IoT devices and services. The majority of these works neither consider properties, such as security and privacy, in their composition methodology nor present a way to verify said properties. A classification of such works is presented in Asghari et al. (2018), where the approaches are divided based on the criterion of the composition method. Said criterion can be the energy consumption, exchanged data, or the IP of the services. However, no security or privacy constraints are considered in the service composition process.

Still, there are some noteworthy approaches that do support description of non-functional IoT service composition properties. Seeger et al. (2018) proposes the offering selection rules (OSRs) that make the reconfiguration of the system possible during runtime. Said rules express non-functional properties that a device/service needs to provide to be part of a specific composition. A possible extension of these rules could include

circularity and trustworthiness-related properties. The work in Lecue and Mehandjiev (2010) introduces an approach for web service composition that uses both semantic and non-functional criteria (QoS) to evaluate the corresponding composition quality. Since non-functional criteria are part of the evaluation of the service composition quality, an extension could focus on circularity and trustworthiness-related criteria. Moreover, Alrifai et al. (2012) present A solution to the QoS-based service composition problem. The existence of quantitative non-functional properties of web services in the way they compose services could be considered as the first step to the circularity and trustworthiness-related properties adoption.

Nevertheless, even IoT service compositions that take under consideration pertinent non-functional requirements (e.g., security or privacy properties) have some other limitations such as Papoutsakis et al. (2021b): (a) lack of automated service selection mechanisms based on those properties; (b) distinction between properties of the individual services and those of the whole service composition and the relationships between the two; and (c) no runtime adaptation of the orchestrations, such as replacement of a component. These deficiencies, albeit focusing only on SPDI properties, have been addressed in the H2020 project SEMIoTICS[3], which provided the baseline implementation for CIRCE's PoC in terms of SPDI properties' verification and reasoning mechanisms.

Methods

Pattern Language Definition

The overall objective of the proposed framework is to be capable to manage IoT systems/ orchestrations based on specified properties as reasoned through patterns. For that reason, it is necessary to develop a language for specifying the components that constitute such IoT orchestrations, their interfaces and interactions, along with the circularity and trustworthiness-related properties that may be required of such components and their orchestrations. More specifically, the language should:

- provide constructs for expressing dependencies between properties at the component and at the orchestration level;
- be structural; It does not prescribe exactly how the functions should be executed nor, e.g., how the ports ensure communication;
- allow for the static and dynamic verification of circularity and trustworthiness-related properties;

A corresponding IoT orchestration model must be defined to provide the foundations for said language specification. A model with the needed characteristics will effectively serve as a general "architecture and workflow model" of the IoT application. Once defined, this model will be used in conjunction with patterns to enable the reasoning required for determining the applicability of particular patterns in specific IoT applications and subsequently reason based on them to enable the different types of adaptation Thus, said model enables us to verify an IoT system/orchestration that satisfies certain properties, or to generate orchestrations that given properties are guaranteed.

[3]Available online at: https://www.semiotics-project.eu/.

The process for defining the system model and specifying the derived language is presented in the subsections that follow.

System Modeling

The original version of the IoT orchestration model (Fysarakis et al., 2019) was developed within SEMIoTICS, where the overall objective was the development of a pattern-driven approach for composing IoT systems/orchestrations with SPDI properties that are guaranteed (Papoutsakis et al., 2020). A brief overview of the model is presented herein, including original model classes, along with additional classes and attributes dedicated to expressing the LCA properties needed to support CIRCE.

In more detail, according to the developed model, the IoT systems/orchestrations are decomposed into individual "placeholders," which implement one or more "activities." The main classes of the model, which are used in the pattern specification, include *Placeholder*, *Orchestration*, *Orchestration Activity*, and *Property*. Placeholders act as predefined positions for different IoT application components. *OrchestrationActivity* class and its subclasses (*IoTSensor, IoTActuator, IoTGateway, SoftwareComponent, NetworkComponent, LinkedActivity*, and *UnAssignedActivity*) are used for the description of said components. Each one of those is able to describe the unique characteristics of the corresponding components in the form of attributes. The *UnAssignedActivity* class makes the model parametric since there is no need for explicitly specifying a specific placeholder. Regarding the orchestration of a set of given Placeholders, it depends on the order in which the corresponding IoT orchestration components' activities must be executed. As a result, an Orchestration can be defined as *Sequence*, *Merge*, *Choice*, *Parallel*, or *Split* (subclasses of the Orchestration class).

Moreover, Properties characterize placeholders, expressing circularity and trustworthiness-related requirements. The state of a *Property* can be required or confirmed. A required property of a placeholder is a property that must be guaranteed for said placeholder to be considered part of a given orchestration with a corresponding property requirement. On the other hand, a confirmed property is a property that is verified at runtime. *Verification* is a class that describes the way a property of a placeholder is verified. The verification process can be conducted through monitoring, testing, a certificate, or via a pattern. For example, a monitoring service could justify that a service or a device is available at specific time windows, in case the desirable property is availability. Moreover, a repository with certificates could give a justification for a property of a placeholder. In case of a pattern, the Mean of verification is the pattern itself; in all the other cases, an interface is needed to a corresponding monitoring tool, testing service, or certificate repository through which the verification can take place.

To express Circularity properties, additional attributes were added to the Placeholder class. These attributes refer to the three primary (*Location, Condition, Availability*) and two operational (*Description, Capability*) intelligent assets properties. **Figure 1** depicts the updated version of the Placeholder class with the whole set of its attributes. As it can be seen, Placeholder is a subclass of the more general class *PropertySubject*. In that way properties can be assigned to a placeholder. Location attribute

refers to the physical, geographical location of an IoT device that is bind with a placeholder. Condition declares the state of a placeholder regarding its lifecycle. Availability can be limited to one of the three values: available, in-use, out-of-order. Moreover, the description attribute includes characteristics necessary for the circular use of the device. More specifically, the description of an IoT device should capture its hardware profile. Finally, the capability attribute lists the different functions or services of a device. One of them is the primary one, while the rest refer to secondary uses of the device. To capture the necessary information of a capability a *Capability* class was created, depicted also in **Figure 1**.

Language Constructs

The described model of the previous section allows for definition of activities along with corresponding control flow operations. In that way, we are able to describe complex IoT orchestrations associating properties of the whole orchestrations with properties of individual components. A language has been created as a product of said model that, except from the description of the IoT orchestrations and the corresponding properties, fulfills the need for static and dynamic property verification and runtime adaptation due to automate process ability.

The language's constructs have been described using Extended Backus–Naur Form (EBNF[4]) grammar. EBNF is a metalanguage used to specify the grammar for a language with precise structure. The original version of the language has been updated due to the newly added LCA properties. An excerpt of the EBNF grammar of the pattern language is presented in **Listing 1**.

```
grammar EBNF;
placeholder
    :    placeholdertitle OPEN_PAREN placeholderid COMMA location
         COMMA condition
COMMA availability COMMA description COMMA capability CLOSE
         _PAREN
    |    orchestration
    |    orchestrationactivity
    ;

orchestration
    :    sequence
    |    parallel
    |    choice
    |    merge
    |    iterate
    |    split
    ;

orchestrationactivity
    :    linkedactivity
    |    unassignedactivity
    |    softwareservice
    |    softwarecomponent
    |    networkcomponent
    |    iotsensor
    |    iotactuator
    |    iotgateway
    |    host
    ;

property
    :    propertytittle OPEN_PAREN propertyname COMMA propertytype
         COMMA category COMMA value COMMA datastate COMMA
         verification COMMA subject COMMA satisfied CLOSE_PAREN
    ;

verification
    :    verificationtitle OPEN\_PAREN verificationtype COMMA means
         CLOSE\_PAREN
    ;
```

Listing 1 | An excerpt of the IoT orchestration language EBNF grammar.

What this excerpt depicts is the definition of some of the most important classes of the IoT orchestration model. The first class that is mentioned is that of a *Placeholder*. The description of a placeholder includes its title (*placeholdertitle*) and then all the attributes necessary for the verification of the circular property (*location, condition, availability, description, capability*). The pipe sign ("|") is used to describe different alternatives of a construct. As shown in the class diagram above, Placeholder class has two subclasses, *orchestration* and *orchestrationactivity*. As a result, a placeholder can be defined as *orchestration* or *orchestrationactivity* alternatives.

An orchestration can be described as one of the different types of orchestrations that the IoT orchestration model allows for, defining the order in which the activities of the corresponding IoT orchestration components must be executed. Moreover, an *orchestrationactivity* is described as one of the allowed categories of IoT orchestration components. The ones mentioned herein are *IoTSensor, IoTActuator, IoTGateway, SoftwareComponent, NetworkComponent, LinkedActivity*, and *UnAssignedActivity*.

Moreover, a lexer and a parser have been created utilizing ANother Tool for Language Recognition (ANTLR[5]), which is a parser generator that allows for reading, processing, and executing structured text and binary files. The ANTLR4 lexer recognizes keywords in any input created with the pattern language transforming them into tokens. ANTLR4 parser uses the created tokens to construct the parse tree, a logical structure. In this way, any input can be checked for compliance with the defined grammar.

Pattern Specification

In this section a formal way for the patterns to be defined is presented. This is another step toward the automated pattern-driven composition of IoT systems/orchestrations where desired properties are guaranteed.

A pattern consists of four parts: (a) the Activity Properties (AP) part that represents the properties of the individual placeholders of a system/orchestration; (b) the Orchestration (ORCH) part that represents the abstract form of the orchestration that the pattern applies to; (c) the Conditions part that describes requirements, constraints, and reactions of the system/orchestration to specific inputs; and (d) the Orchestration Properties (OP) part that represents the orchestration-level properties that the pattern can guarantee for the orchestration specified in the ORCH part.

[4]Available online at: https://tomassetti.me/ebnf/.

[5]Available online at: https://www.antlr.org/.

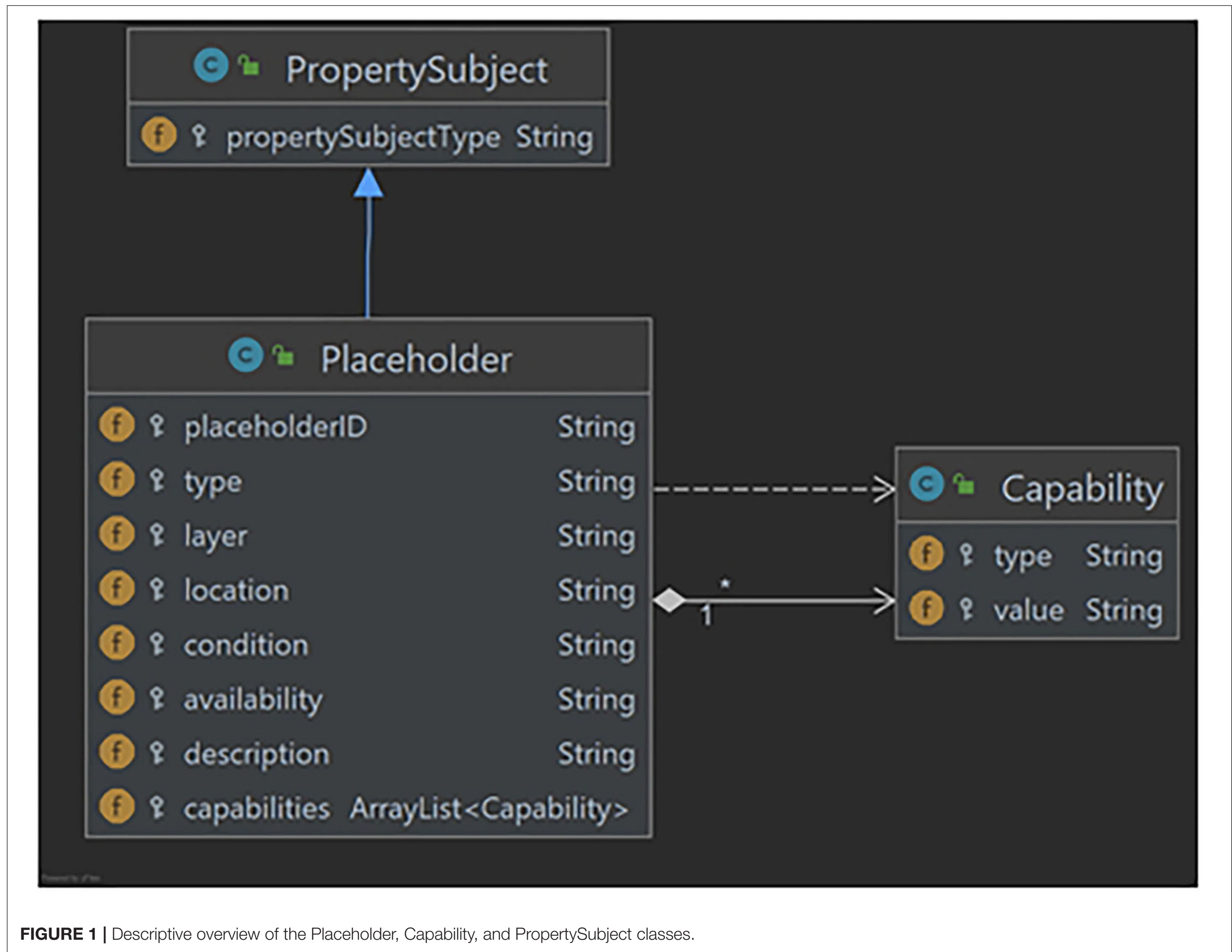

FIGURE 1 | Descriptive overview of the Placeholder, Capability, and PropertySubject classes.

The described structure of a pattern in the form of a formula is expressed as:

$$AP \wedge ORCH \wedge CONDITIONS \models OP$$

If the AP properties of an orchestration placeholders and the conditions of the pattern hold, then the OP property specified in the pattern also holds for the whole ORCH.

APs can be described using the Property class described above. Property name uniquely identifies the property and the Property Subject depicts the placeholder that implements the activity for which the property is required or verifiable (*propertytype*). ORCH is an object of Orchestration class including placeholder instances. Conditions are materialized using the Operation and Parameters classes. Inputs and outputs of the activity placeholders of the pattern are defined in the objects of those two classes. Finally, OP is a Property object referring to the whole orchestration.

Automated Pattern Reasoning

Due to the need for automated processing and management of the defined patterns, Drools[6] is selected as a mean for expressing those patterns in the form of machine processable business production rules. Drools is a business rules management system (BRMS) solution and allows for the construction, maintenance, and enforcement of business policies in an organization, an application, or a service. The Drools rules have the following structure:

rule name <attributes>*
when <conditional element>*
then <action>* **end**

The match between the Facts in the Drools Knowledge Base (KB) and the conditions expressed in the **when** part of the rule, is the way a Drools rule is applied. The execution of the actions in the **then** part is what follows. These actions insert, retract or update facts in the KB using the corresponding standard

[6] Available online at: https://www.drools.org/.

Drools actions. The conditional elements are used to define constraints for the data in the KB. Said constraints can be simple or complex utilizing logical operators such as *and, or, not, exists, forall, contains.*

The aforementioned ANTLR4 lexer and parser create a Drools fact for every orchestration activity, control flow operation and property. The Drools facts are then inserted in the Drools KB, where all the knowledge definitions live. Knowledge sessions, created from the KB, allow interaction between Drools and the core component to fire Drools rules and perform reasoning. Rules themselves are also hold in a knowledge session.

Patterns corresponding to each of the circularity and trustworthiness-related properties are presented here as showcases of how patterns are translated to Drools rules.

RESULTS

The CIRCE Architecture

To implement the CIRCE approach, a number of building blocks need to be designed, developed and deployed across the various layers of an IoT system (namely, Backend, 5G network and Edge). **Figure 2** depicts the resulting high-level architecture of the CIRCE approach. Key components include:

- **Pattern Orchestrator**: This module features a semantic reasoner able to understand instantiated IoT orchestrations [e.g., the "Recipes" approach, based on NODE-RED[7] (Thuluva et al., 2017; Papoutsakis et al., 2020)] and transform them into composition structures to be used by architectural patterns to guarantee the required properties. The Pattern Orchestrator is then responsible to pass said patterns to the corresponding Pattern Engines (as defined in the Backend, Network and Edge layers), selecting for each of them the subset of these that refer to components under their control (e.g., passing Network -specific patterns to the Pattern Module present in the 5G SDN controller).
- **Backend Pattern Engine**: This module enables the capability to insert, modify, execute and retract patterns at design or at runtime in the backend; these interactions happen through interfacing with the Pattern Orchestrator (see above). Moreover, it is able to reason on the specified properties at a local (backend) and global level. To enable the latter, the Backend Pattern Engine is able to receive fact updates from the individual Pattern Engines present at the lower layers (Network & Field), allowing it to have an up-to-date view of the state of said layers and their corresponding components.
- **Network Pattern Engine**: Integrated in the SDN controller to enable the capability to insert, modify, execute and retract network-level patterns at design or at runtime. It provides the capability to reason on the specified properties at the network layer. From an implementation perspective, it is supported by the integration of all required dependencies within the network controller (e.g., with the path manager and resource manager of the controller), as well as the interfaces allowing entities that interact with the controller to be managed based on desired patterns at design and at runtime. It features different subcomponents as required by the rule engine, such as the knowledge base, the core engine and the compiler.
- **Edge Pattern Engine**: Typically deployed on the IoT/IIoT gateway, able to host patterns as provided by the Pattern Orchestrator. Since the compute capabilities of the gateway can be limited, the module is able to host patterns in an executable form compared to the pattern rules as provided in the other layers. The executable patterns are able to guarantee desired properties locally (i.e., gateway and IoT edge devices monitored by that gateway).

As is evident from the architecture, a key design decision was to have separate reasoning engines for each of the three layers considered (Backend, Network and Edge). This way, reasoning engines that may have strict limitations in resources and/or response time (e.g., within the network or at the edge), only receive and have to reason on properties affecting the specific layer, thus having more efficient operation (less memory to store rules, less reasoning time, quicker response time to needed adaptations). Furthermore, this allows the individual layers to operate and reason autonomously, post-deployment, without requiring interactions with the backend for that task. Then, only the backend layer (where resource limitations are typically not an issue) keeps the global view and reasons for the global (and end-to-end) properties of a specific orchestration.

CSPDI and LCA Patterns

An essential element in the operation of CIRCE are the CSPDI and LCA patterns themselves. Therefore, this section presents a first set circularity and trustworthiness-related patterns, covering all key properties, in the form of Drools rules. As already mentioned, the Drools rules need Facts in the KB to be matched with the conditions in the **when** part of the rules. Said Facts are no other than description of IoT orchestrations along with their properties. Such an orchestration is received as input, expressed with the IoT orchestration language described in section Language Constructs above.

Connectivity

The Connectivity property specified herein allows for the establishment of a QoS-enabled connection between declared end-points. As CIRCE is designed to operate in the context of a 5G-enabled IoT infrastructure, the existence of a Software Defined Network (SDN) Controller is assumed. This allows CIRCE to exploit the enhanced flexibility and design- and run-time adaptations offered by SDN networks [e.g., *via* Service Function Chaining to ensure desired properties are maintained at runtime (Petroulakis, 2018)].

Upon a connectivity request, the corresponding path between the two end-points is computed and the QoS constraints are defined in the form of flow rules and queue mapping (*via* Path Manager of SDN controller). As soon as the defined flow rules are installed (*via* Resource Manager of SDN controller), the end-point connectivity is enabled (SEMIoTICS, 2020).

The input orchestration is a sequence of two placeholders connected with a link. This link is not the physical link

[7] Available online at: https://nodered.org/.

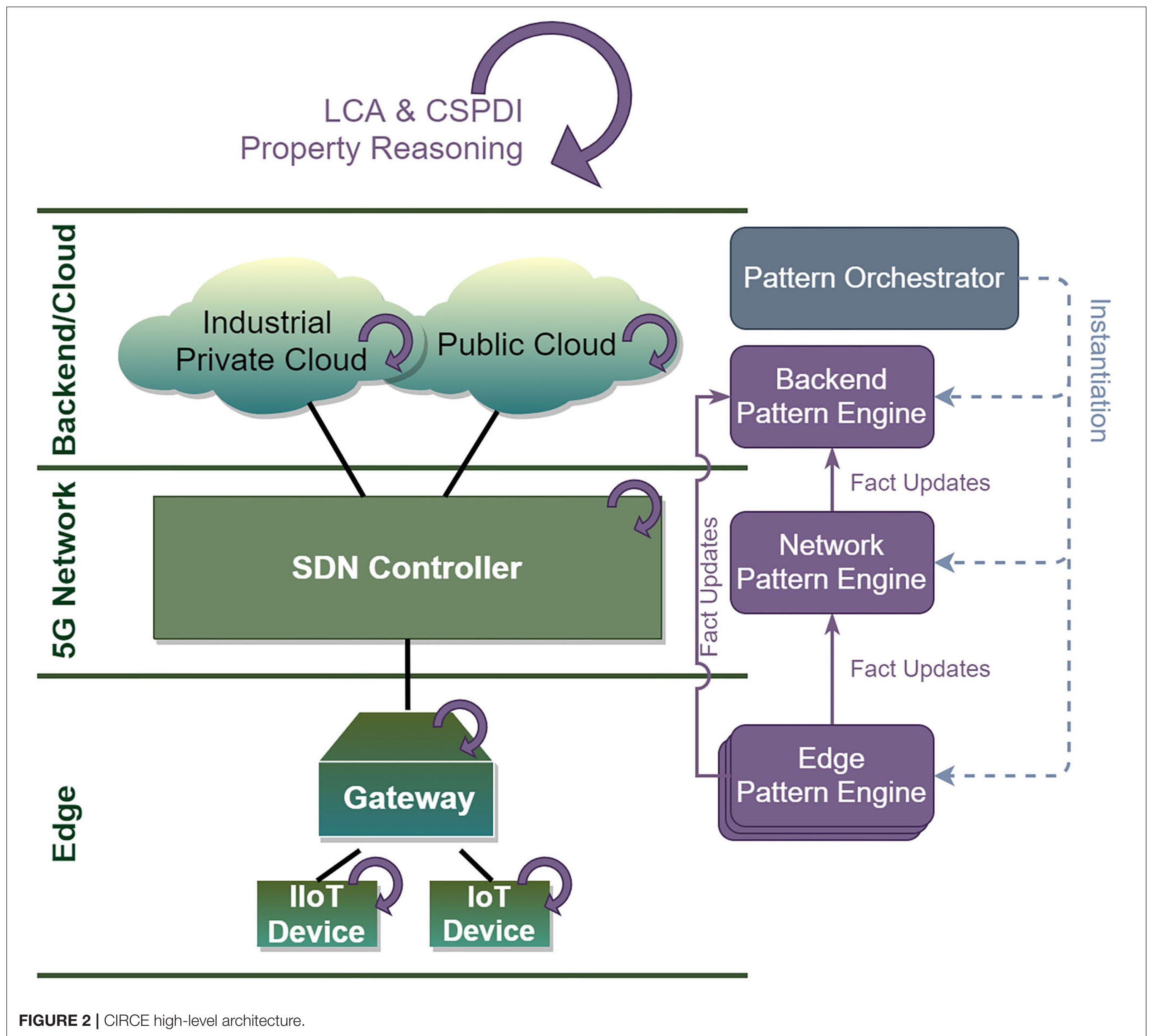

FIGURE 2 | CIRCE high-level architecture.

that eventually will connect the two placeholders; it just represents the data flow between two components of the IoT orchestration. Additionally, the desired QoS properties of the connectivity are described, namely *PathBandwidth*, *PathDelay*, *PathBurst*, and *PathResilience*. In this example IoT orchestration description, said properties are described with given corresponding values (*reqBwKbps, reqDelayMs, reqBurstKbps, resilience*), and as already satisfied (*satisfied=true*). On the other hand, the satisfied attribute of the property to be validated (*Connectivity*) is set to false. The orchestration description is as follows:

1. **ORCH "Connectivity"**
2. Placeholder (P1)
3. Placeholder (P2)
4. Link (L1, P1, P2)
5. Sequence (S1, P1, P2, L1)
6. Property (Pr1, category=PathBandwidth, subject=S1, value=reqBwKbps, satisfied=true)
7. Property (Pr2, category=PathDelay, subject=S1, value=reqDelayMs, satisfied=true)
8. Property (Pr3, category=PathBurst, subject=S1, value=reqBurstKbps, satisfied=true)
9. Property (Pr4, category=PathResilience, subject=S1, value=resilience, satisfied=true)
10. Property (Pr5, category=Connectivity, subject=S1, satisfied=false)

To establish the QoS-enabled connectivity between declared orchestration placeholders, we rely on instantiation of the

relevant Connectivity pattern, expressed as Drools rule in **Listing 2** below.

```
rule ``Connectivity Pattern''
when
  Placeholder($sensor1:=placeholderid1, $srcMac:=MAC1, $srcIp:=
      ipAddress1)
  Placeholder($sensor2:=placeholderid2, $dstMac:=MAC2, $dstIp:=
      ipAddress2)
  Link($link1:=linked, $sensor1:=placeholdera, $sensor2:=placeholderb)
  Sequence($seq1:=placeholderid, $sensor1:=placeholdera, $sensor2:=
      placeholderb)
  $PR1: Property ($seq1:=subject, category==``PathBandwidth,''
      $reqBwKbps:=value, satisfied==true)
  $PR2: Property ($seq1:=subject, category==``PathDelay,'' $reqDelayMs:=
      value, satisfied==true)
  $PR3: Property ($seq1:=subject, category==``PathBurst,'' $reqBurstKbps
      :=value, satisfied==true)
  $PR4: Property ($seq1:=subject, category==``PathResilience,'' $resilience
      :=value, satisfied==true)
  $PR5: Property ($seq1:=subject, category==``Connectivity,'' satisfied==
      false)
then
  modify($PR5){satisfied=true}
  try {
    RefMonProxy.applicationAddRequest( $srcMac,
                                       $dstMac,
                                       $srcIp,
                                       $dstIp,
                                       $reqBwKbps.longValue(),
                                       $reqDelayMs.longValue(),
                                       $reqBurstKbps.longValue(),
                                       $resilience.longValue());
  } catch(Exception ex) {
  System.out.printIn(ex.getStackTrace());
  }
end
```

Listing 2 | Connectivity pattern in the form of Drools rule.

The **when** part of the rule specifies: (a) two placeholders that are parts of a given orchestration, (b) a link between them, (c) the type of the orchestration that is a Sequence in this case, (d) four QoS properties, and (e) the overall orchestration property that can be guaranteed through the application of the pattern.

The **then** part triggers the enforcement of the QoS properties which uses an SDN controller to generate in the switches per flow mapping configurations. This is done by calling the *RefMonProxy.applicationAddRequest* method, in this case. The request properties considered in the instantiation of the network service are: (a) the required bandwidth share in Kilobits per Second (Kbps); (b) the requested end-to-end delay requirement in milliseconds; (c) the input traffic burst in max. Kbps; (d) the source MAC address; (e) the destination MAC Address; (f) the requirement for resilient path establishment.

After establishment of the above flow, the referenced endpoints (with given MAC addresses as identifiers in the flow rules) are guaranteed the requested QoS requirements (bandwidth and delay). This is assuming that that the input traffic arrivals are shaped as per promised maximal traffic burst and sending rate and do not exceed the requested rate.

Security

Security patterns are presented in detail in Papoutsakis et al. (2021a), in the form of a hierarchical taxonomy of properties. All properties are mapped to corresponding so called "high level" and "low level" patterns. The former are solutions for high level problems and create the context for the latter, which are considered more specific and include practical guidelines solving well-defined problems. Security is typically decomposed to Confidentiality, Integrity, and Availability (Stallings et al., 2012). The IoT orchestration that is used as input consists of a sequence of two placeholders, connected by a link and is presented using the IoT orchestration language below:

11. **ORCH "Security"**
12. Placeholder (P1)
13. Placeholder (P2)
14. Link (L1, P1, P2)
15. Sequence (S1, P1, P2, L1)
15. Property (Pr1, category=Security, subject=S1, satisfied=false)

The last line of the description describes the property to be verified (=Security property). Based on the above, the corresponding Drools rule is depicted in **Listing 3**.

```
rule ``Security Pattern''
when
  Placeholder($sensor1:=placeholderid1)
  Placeholder($sensor2:=placeholderid2)
  Link($link1:=linked, $sensor1:=placeholdera, $sensor2:=placeholderb)
  Sequence($seq1:=placeholderid, $sensor1:=placeholdera, $sensor2:=
      placeholderb)
  $PR: Property ($seq1:=subject, category==``Security,'' satisfied==false)
then
  Property s1Property = new Property();
  s1Property.setCategory(``Confidentiality'');
  s1Property.setSubject($seq1);
  s1Property.setSatisfied(false);
  insert(s1Property);

  Property s2Property = new Property();
  s2Property.setCategory(``Interoperability'');
  s2Property.setSubject($seq1);
  s2Property.setSatisfied(false);
  insert(s2Property);

  Property s3Property = new Property();
  s3Property.setCategory(``Availability'');
  s3Property.setSubject($seq1);
  s3Property.setSatisfied(false);
  insert(s3Property);
end
```

Listing 3 | Security pattern in the form of Drools rule.

The orchestration (ORCH) that is chosen here is a sequence of two placeholders, as can be seen in the **when** part of the rule. The last line of this part of the rule declares the OP property of the pattern in question. In the **then** part, three new properties are created (Confidentiality, Integrity, and Availability), which correspond to the AP properties of the pattern, which in this case are referred to the whole orchestration. According to the pattern specification presented earlier, if the Confidentiality, Integrity,

and Availability properties of the orchestration hold, then the Security property specified in the pattern also holds for the whole orchestration.

The tree of the hierarchical taxonomy of properties continues with more layers decomposing Confidentiality, Integrity, and Availability to more properties corresponding to lower level patterns. Confidentiality is broken down to Encrypted Channel, Encrypted Storage and Encrypted Processing; Integrity to Safe Channel, Safe Storage and Safe Processing; and Availability to Uptime, Redundancy, and Fault Management.

Confidentiality

Every one of the decompositions, described in the previous subsection, represents relationships among properties, corresponds to a pattern and can be described by a Drools rule. For example, the Drools rule for Confidentiality is depicted in **Listing 4**. But before presenting the actual rule, the IoT orchestration used as input is once again a sequence of two placeholders, connected by a link:

1. **ORCH "Confidentiality"**
2. Placeholder (P1)
3. Placeholder (P2)
4. Link (L1, P1, P2)
5. Sequence (S1, P1, P2, L1)
6. Property (Pr1, category=Confidentiality, subject=S1, satisfied=false)

The difference with the ORCH Security is in the last line where a Confidentiality property is described as the property to be verified.

```
rule ``Confidentiality Pattern''
when
  Placeholder($sensor1:=placeholderid1)
  Placeholder($sensor2:=placeholderid2)
  Link($link1:=linked, $sensor1:=placeholdera, $sensor2:=placeholderb)
  Sequence($seq1:=placeholderid,$sensor1:=placeholdera,$sensor2:=
       placeholderb)
  $PR: Property ($seq1:=subject, category==``Confidentiality,'' satisfied==
       false)
then
  Property s1Property = new Property();
  s1Property.setCategory(``Encrypted Storage'');
  s1Property.setSubject($sensor1);
  s1Property.setSatisfied(false);
  insert(s1Property);
  Property s2Property = new Property();
  s2Property.setCategory(``Encrypted Processing'');
  s2Property.setSubject($sensor1);
  s2Property.setSatisfied(false);
  insert(s2Property);

  Property s3Property = new Property();
  s3Property.setCategory(``Encrypted Storage'');
  s3Property.setSubject($sensor2);
  s3Property.setSatisfied(false);
  insert(s3Property);
  Property s4Property = new Property();
  s4Property.setCategory(``Encrypted Processing'');
  s4Property.setSubject($sensor2);
  s4Property.setSatisfied(false);
  insert(s4Property);

  Property s5Property = new Property();
  s5Property.setCategory(``Encrypted Channel'');
  s5Property.setSubject($link1);
  s5Property.setSatisfied(false);
  insert(s5Property);
end
```

Listing 4 | Confidentiality pattern in the form of Drools rule.

The orchestration (ORCH) is again a sequence of two placeholders. The last line of the **when** part of the rule declares the OP property of the pattern in question. In the **then** part, three new properties are created (Encrypted Channel, Encrypted Storage, and Encrypted Processing), which correspond to the AP properties of the pattern. Encrypted Processing and Encrypted Storage are assigned to the two placeholders of the orchestration, while the Encrypted Channel is assigned to the link between them. If the Encrypted Channel, Encrypted Storage, and Encrypted Processing properties hold for the corresponding components of the orchestration, then the Confidentiality property specified in the pattern also holds for the whole orchestration.

Integrity

A very similar IoT orchestration is also used as input for the demonstration of the verification of the Integrity property, and a very similar rule to express its decomposition to the Safe Channel, Safe Storage and Safe Processing properties.

1. **ORCH "Integrity"**
2. Placeholder (P1)
3. Placeholder (P2)
4. Link (L1, P1, P2)
5. Sequence (S1, P1, P2, L1)
6. Property (Pr1, category=Integrity, subject=S1, satisfied=false)

The rule is shown in **Listing 5**.

```
rule ``Integrity Pattern''
when
  Placeholder($sensor1:=placeholderid1)
  Placeholder($sensor2:=placeholderid2)
  Link($link1:=linked, $sensor1:=placeholdera, $sensor2:=placeholderb)
  Sequence($seq1:=placeholderid,$sensor1:=placeholdera,$sensor2:=
       placeholderb)
  $PR: Property ($seq1:=subject, category==``Integrity,'' satisfied==false)
then
  Property s1Property = new Property();
  s1Property.setCategory(``Safe Storage'');
  s1Property.setSubject($sensor1);
  s1Property.setSatisfied(false);
  insert(s1Property);
  Property s2Property = new Property();
  s2Property.setCategory(``Safe Processing'');
  s2Property.setSubject($sensor1);
  s2Property.setSatisfied(false);
  insert(s2Property);

  Property s3Property = new Property();
  s3Property.setCategory(``Safe Storage'');
  s3Property.setSubject($sensor2);
  s3Property.setSatisfied(false);
  insert(s3Property);
  Property s4Property = new Property();
```

```
  s4Property.setCategory(``Safe Processing'');
  s4Property.setSubject($sensor2);
  s4Property.setSatisfied(false);
  insert(s4Property);

  Property s5Property = new Property();
  s5Property.setCategory(``Safe Channel'');
  s5Property.setSubject($link1);
  s5Property.setSatisfied(false);
  insert(s5Property);
end
```

Listing 5 | Integrity pattern in the form of Drools rule.

The orchestration is a sequence of two placeholders, the OP property is Integrity, and the AP properties are the Safe Channel, Safe Storage and Safe Processing properties. Similarly to the previous rule, If the Safe Channel, Safe Storage, and Safe Processing properties hold for the corresponding components of the orchestration, then the Integrity property also holds for the whole orchestration.

Availability

Following the same norm with the previous examples, the input IoT orchestration is a sequence of two placeholders and the Drools rule of Availability defines that if the Uptime, Redundant Storage, and Fault Management properties hold for an orchestration, then the Integrity property also holds for the same orchestration.

1. **ORCH "Availability"**
2. Placeholder (P1)
3. Placeholder (P2)
4. Link (L1, P1, P2)
5. Sequence (S1, P1, P2, L1)
6. Property (Pr1, category=Availability, subject=S1, satisfied=false)

The corresponding Drools rule is shown in **Listing 6**.

```
rule ``Availability''
when
  Placeholder($sensor1:=placeholderid1)
  Placeholder($sensor2:=placeholderid2)
  Link($link1:=linked, $sensor1:=placeholdera, $sensor2:=placeholderb)
  Sequence($seq1:=placeholderid, $sensor1:=placeholdera, $sensor2:=
       placeholderb)
  $PR: Property ($seq1:=subject, category==``Availability,'' satisfied==false)
then
  Property s1Property = new Property();
  s1Property.setCategory(``Uptime'');
  s1Property.setSubject($seq1);
  s1Property.setSatisfied(false);
  insert(s1Property);

  Property s2Property = new Property();
  s2Property.setCategory(``Redundant Storage'');
  s2Property.setSubject($seq1);
  s2Property.setSatisfied(false);
  insert(s2Property);

  Property s3Property = new Property();
  s3Property.setCategory(``Fault Management'');
  s3Property.setSubject($seq1);
  s3Property.setSatisfied(false);
  insert(s3Property);
end
```

Listing 6 | Availability pattern in the form of Drools rule.

Privacy

Although, there is a lack of taxonomy (Caiza et al., 2017) based on the consensus of works in the area (Ahituv et al., 1987; Pfitzmann and Hansen, 2010; Kuhn et al., 2019), Privacy is decomposed to eight key privacy concepts namely (i) data protection; (ii) authentication; (iii) authorization; (iv) anonymity; (v) pseudonymity; (vi) unlinkability; (vii) undetectability, and (viii) unobservability.

The input IoT orchestration is:

1. **ORCH "Privacy"**
2. Placeholder (P1)
3. Placeholder (P2)
4. Link (L1, P1, P2)
5. Sequence (S1, P1, P2, L1)
6. Property (Pr1, category=Privacy, subject=S1, satisfied=false)

The corresponding Drools rule for Privacy is depicted in **Listing 7**. According to the **when** part of the rule, the orchestration (ORCH) is once again a sequence of two placeholders and the declared OP property is Privacy. In the **then** part, eight new properties are created, one for each of the privacy concepts the Privacy property is decomposed to (data protection, authentication, authorization, anonymity, pseudonymity, unlinkability, undetectability, and unobservability). These properties correspond to the AP properties of the pattern. If the eight properties hold for the corresponding components of the orchestration, then the Privacy property specified in the pattern also holds for the whole orchestration.

```
rule ``Privacy Pattern''
when
  Placeholder($sensor1:=placeholderid1)
  Placeholder($sensor2:=placeholderid2)
  Link($link1:=linked, $sensor1:=placeholdera, $sensor2:=placeholderb)
  Sequence($seq1:=placeholderid,$sensor1:=placeholdera,$sensor2:=
       placeholderb)
  $PR: Property ($seq1:=subject, category==``Privacy,'' satisfied==false)
then
  Property s1Property = new Property();
  s1Property.setCategory(``Data Protection'');
  s1Property.setSubject($seq1);
  s1Property.setSatisfied(false);
  insert(s1Property);
  Property s2Property = new Property();
  s2Property.setCategory(``Authentication'');
  s2Property.setSubject($seq1);
  s2Property.setSatisfied(false);
  insert(s2Property);
  Property s3Property = new Property();
  s3Property.setCategory(``Authorization'');
  s3Property.setSubject($seq1);
  s3Property.setSatisfied(false);
  insert(s3Property);
  Property s4Property = new Property();
  s4Property.setCategory(``Anonymity'');
  s4Property.setSubject($seq1);
```

```
  s4Property.setSatisfied(false);
  insert(s4Property);
  Property s5Property = new Property();
  s5Property.setCategory(``Pseudonymity'');
  s5Property.setSubject($seq1);
  s5Property.setSatisfied(false);
  insert(s5Property);
  Property s5Property = new Property();
  s5Property.setCategory(``Unlinkability'');
  s5Property.setSubject($seq1);
  s5Property.setSatisfied(false);
  insert(s5Property);
  Property s5Property = new Property();
  s5Property.setCategory(``Undetectability'');
  s5Property.setSubject($seq1);
  s5Property.setSatisfied(false);
  insert(s5Property);
  Property s5Property = new Property();
  s5Property.setCategory(``Unobservability'');
  s5Property.setSubject($seq1);
  s5Property.setSatisfied(false);
  insert(s5Property);
end
```

Listing 7 | Privacy pattern in the form of Drools rule.

Dependability

According to Papoutsakis et al. (2021a), Dependability focuses on reliability, fault tolerance and safety. An IoT orchestration consisting of a merge of two IoT-sensors is the most appropriate orchestration to showcase the Dependability pattern.

1. **ORCH "Dependability"**
2. IoTSensor (S1)
3. IoTSensor (S2)
4. Gateway (G1)
5. Link (L1, P1, G1)
6. Link (L2, P2, G1)
7. Merge (M1, S1, S2, G1, L1, L2)
7. Property (Pr1, category=Dependability, subject=M1, satisfied=false)

The Drools rule for dependability would look like the one in **Listing 8**.

```
rule ``Dependability''
when
  IoTSensor($sensor1:=placeholderid)
  IoTSensor($sensor2:=placeholderid)
  Merge($merge1:=placeholderid, $sensor1:=placeholdera, $sensor2:=
        placeholderb)
  $PR: Property ($merge1:=subject, category== ``Dependability,'' satisfied
        ==false)
then
  Property s1Property = new Property();
  s1Property.setCategory(``Reliability'');
  s1Property.setSubject($merge1);
  s1Property.setSatisfied(false);
  insert(s1Property);

  Property s2Property = new Property();
  s2Property.setCategory(``Fault Tolerance'');
  s2Property.setSubject($merge1);
  s2Property.setSatisfied(false);
  insert(s2Property);

  Property s3Property = new Property();
  s3Property.setCategory(``Safety'');
  s3Property.setSubject($merge1);
  s3Property.setSatisfied(false);
  insert(s3Property);
end
```

Listing 8 | Dependability pattern in the form of Drools rule.

As can be seen in the **when** part of the rule, the orchestration (ORCH) is a merge of two IoT sensors and the declared OP property is Dependability. In the **then** part, three new properties are created and assigned to the orchestration (Reliability, Fault Tolerance and Safety), describing the AP properties of the pattern. If the newly created properties hold for the orchestration, then the Dependability property specified in the pattern also holds for the whole orchestration.

Interoperability

According to Hatzivasilis et al. (2018), there are different types of interoperability named technical, syntactic, semantic, and organizational interoperability. The former makes the cooperation of heterogeneous devices, which use different communication protocols, possible. The syntactic type of interoperability determines well-defined data formats, interfaces, and encoding. Semantic interoperability defines data models and ontologies that are commonly accepted among the heterogeneous devices. Finally, the organizational type allows for integration and orchestration of services that reside in different domains and platforms.

It should also be mentioned that latter levels of interoperability assume the existence of former ones to be achieved. For example, technical interoperability is prerequisite for achieving syntactic interoperability. A sequence of two placeholders is once again the input IoT orchestration:

1. **ORCH "OrganizationalInteroperability"**
2. Placeholder (P1)
3. Placeholder (P2)
4. Link (L1, P1, P2)
5. Sequence (S1, P1, P2, L1)
6. Property (Pr1, category= OrganizationalInteroperability, subject=S1, satisfied=false)

Based on the described classification of Interoperability, the corresponding Drools rules is depicted in **Listing 9**.

```
rule ``Organizational interoperability''
when
  Placeholder($sensor1:$placeholderid1)
  Placeholder($sensor2:$placeholderid2)
  Link($link1:$linked, $sensor1:$placeholdera, $sensor2:$placeholderb)
  Sequence($seq1:$placeholderid,$sensor1:$placeholdera,$sensor2:
        $placeholderb)
  $PR: Property ($seq1:$subject, category==``Organizational interoperability,
        '' satisfied==false)
then
  Property s1Property $ new Property();
  s1Property.setCategory(``Semantic interoperability'');
  s1Property.setSubject($seq1);
  s1Property.setSatisfied(false);
  insert(s1Property);
end
```

```
rule ``Semantic interoperability''
when
  Placeholder($sensor1:$placeholderid1)
  Placeholder($sensor2:$placeholderid2)
  Link($link1:$linked, $sensor1:$placeholdera, $sensor2:$placeholderb)
  Sequence($seq1:$placeholderid,$sensor1:$placeholdera,$sensor2:
       $placeholderb)
  $PR: Property ($seq1:$subject, category==``Semantic interoperability,''
       satisfied==false)
then
  Property s1Property $ new Property();
  s1Property.setCategory(``Syntactic interoperability'');
  s1Property.setSubject($seq1);
  s1Property.setSatisfied(false);
  insert(s1Property);
end

rule ``Syntactic interoperability''
when
  Placeholder($sensor1:$placeholderid1)
  Placeholder($sensor2:$placeholderid2)
  Link($link1:$linked, $sensor1:$placeholdera, $sensor2:$placeholderb)
  Sequence($seq1:$placeholderid,$sensor1:$placeholdera,$sensor2:
       $placeholderb)
  $PR: Property ($seq1:$subject, category== ``Syntactic interoperability,''
       satisfied==false)
then
  Property s1Property $ new Property();
  s1Property.setCategory(``Technical interoperability'');
  s1Property.setSubject($seq1);
  s1Property.setSatisfied(false);
  insert(s1Property);
end
```

Listing 9 | Interoperability pattern in the form of Drools rule.

As it is mentioned above, the latter levels of interoperability assume the existence of former ones. This is what the three rules in **Listing 9** expresses. According to the first one, if the Organizational Interoperability property is to be verified for a given sequence of placeholders, the Semantic Interoperability property needs to be verified first. This is why such a property is created in the **then** part of the rule and is assigned to the given sequence.

The same holds, depicted by the second rule, for the Semantic Interoperability property and its prerequisite the Syntactic Interoperability. If the property to be verified is a Semantic Interoperability property (**when** part of the rule), a new Syntactic Interoperability property is created and assigned to the given orchestration (**then** part of the rule). Finally, the exact same relationship between the Syntactic Interoperability property and the Technical Interoperability is depicted in the last rule.

Circularity Properties (LCA)

There are two domains or planes in which circularity is defined: a) the cyber plane that refers to the ICT infrastructure that provides resources regarding computing, networking and connectivity, and b) the intelligent assets plane involving the interconnection and interaction of the actors who are placed on a physical space. Properties that can be used regarding the intelligent assets plane, which is our main focus, are Location, Condition and Availability.

The input IoT orchestration is a merge of two sensors as shown below:

1. **ORCH "Circularity"**
2. IoTSensor (S1)
3. IoTSensor (S2)
4. Gateway (G1)
5. Link (L1, P1, G1)
6. Link (L2, P2, G1)
7. Merge (M1, S1, S2, G1, L1, L2)
8. Property (Pr1, category=Circularity, subject=M1, satisfied=false)

All the necessary parts of the merge are expressed such as the two sensors, the Gateway where they merge their outputs, and the links among them. Line 8 corresponds to the Circularity property that we want to validate for the given orchestration.

To express the decomposition of the Circularity property, regarding the intelligent assets plane, the Drools rule shown in **Listing 10** is used.

```
rule ``Circularity"
when
   IoTSensor($sensor1:=placeholderid)
   IoTSensor($sensor2:=placeholderid)
   Merge($merge1:=placeholderid, $sensor1:=
     placeholdera, $sensor2:=placeholderb)
   $PR: Property ($merge1:=subject, category==``
     Circularity,'' satisfied==false)
then
   Property s1Property = new Property();
   s1Property.setCategory(``Location'');
   s1Property.setSubject($merge1);
   s1Property.setSatisfied(false);
   insert(s1Property);

   Property s2Property = new Property();
   s2Property.setCategory(``Condition'');
   s2Property.setSubject($merge1);
   s2Property.setSatisfied(false);
   insert(s2Property);

   Property s3Property = new Property();
   s3Property.setCategory(``Availability'');
   s3Property.setSubject($merge1);
   s3Property.setSatisfied(false);
   insert(s3Property);
end
```

Listing 10 | Circularity pattern in the form of Drools rule.

What the Circularity rule actually says is that if the Location, Condition and Availability properties hold for an orchestration, then the Circularity property specified in the pattern also holds for the whole orchestration. Circularity allows for the inclusion of unused or under-used assets during the initialization of a service. Additional Drools rules are presented below including Location, Condition, and Availability properties.

Location

Location refers to the physical location of the component (e.g., the location where an IoT sensor is installed), and it can be expressed as a description of a pair of coordinates. The knowledge of the location of an asset can extend its use cycle length. For

example, it can guide the replacement of a broken component or optimize the route of a vehicle to avoid its wear. Moreover, the utilization of an asset can be increased since the driving time of a vehicle can be reduced, or shared assets can be relocated faster. The knowledge of the location can also automate the localization of goods on secondary markets or allow for reverse logistics planning. Finally, it can help the regeneration of natural capital through the automated tracking of their location.

The Drools rule, in **Listing 11** below, verifies that the Location property holds for an *IoTSensor* if the value of the location attribute is not null. The verification of the fact that the property holds is done by modifying the satisfied attribute to true.

The needed IoT orchestration consists of just a sensor and the property in question, as shown below:

1. **ORCH "Location"**
2. IoTSensor (S1)
3. Property (Pr1, category=Location, subject=M1, satisfied=false)

The satisfied attribute of the property is given as *false*, in order to be changed to *true* if the property actually holds.

```
rule ``Location''
when
  IoTSensor($sensor1:=placeholderid)
  $PR: Property ($sensor1:=subject, category==``Location,'' $location:=
        location, $location!=null, satisfied==false)
then
  modify($PR)\{satisfied=true\};
end
```

Listing 11 | Location pattern in the form of Drools rule.

Condition

As shown in the Circularity rule above, when a Circularity property of an orchestration is to be verified, a Condition property, among others, is created and assigned to the same orchestration. This new property is to be verified, too. Condition can take values such as *good* or *requiring maintenance*, *repair*, *refurbishment*, and *recycling*. The knowledge of the condition of an asset can extend its use cycle length since it allows the prediction of its maintenance or replacement. The utilization is also increased. For example, precise use of the input factors (fertilizer) in agriculture can be achieved. Looping assets can be enhanced through accurate asset assessment.

The rule depicted in **Listing 12** does exactly that; modifying the "satisfied" attribute of the Property class to true (**then** part of the rule). The verification takes place when the value of the condition attribute of the Condition property of an IoTSensor, in this case, is *good* or *n/a* (**when** part of the rule). In practice this means that the condition of an IoTSensor allows for its reuse in an application. If the value of the condition attribute was *requiring maintenance or service, refurbished, or recycled*, the verification could not happen.

The needed IoT orchestration once again consists of just a sensor and the property in question, as shown below:

1. **ORCH "Condition"**
2. IoTSensor (S1)
3. Property (Pr1, category=Condition, subject=M1, satisfied=false)

```
rule ``Condition''
when
  IoTSensor($sensor1:=placeholderid)
  $PR: Property ($sensor1:=subject, category==``Condition,'' condition
        ==``good''|| condition==''n/a,'' satisfied==false)
then
  modify($PR)\{satisfied=true\};
end
```

Listing 12 | Condition pattern in the form of Drools rule.

Availability

Availability is the third property that need to be verified for the Circularity property to be verified. A possible Drools rule expressing the Availability property was presented in Availability, but that referred to availability from a security perspective. Another possible way to express Availability, from a circularity perspective, is presented here. In this context, the acceptable values (sub-properties) identified are: *working*, *ready for reuse*, or *broken*. The knowledge of the availability of an asset can extend its use cycle length. For example, energy systems can optimize sizing, supply and maintenance based on usage patterns. Regarding utilization of assets, the automated connection of available, shared assets can be achieved.

The corresponding rule is depicted in **Listing 13**. The "satisfied" attribute of the Availability Property class is modified to true (**then** part of the rule), if the value of the availability attribute of the Availability property of an IoTSensor, is *ready for reuse* (**when** part of the rule). If the value of the condition attribute was *working or broken*, the verification could not happen.

The needed IoT orchestration once again consists of just a sensor and the property in question, as shown below:

1. **ORCH "Availability"**
2. IoTSensor (S1)
3. Property (Pr1, category=Availability, subject=M1, satisfied=false)

```
rule ``Availability''
when
  IoTSensor($sensor1:=placeholderid)
  $PR: Property ($sensor1:=subject, category==``Availability,'' condition
        ==``ready for reuse,'' satisfied==false)
then
  modify($PR)\{satisfied=true\};
end
```

Listing 13 | Availability pattern in the form of Drools rule.

Evaluation Results

An evaluation based on a PoC implementation of CIRCE was carried out, focusing on assessing the performance and scalability of the automated verification of given circularity and trustworthiness-related properties of an IoT orchestration.

The testbed includes two physical machines. The first one is a backend server (6-core CPU, 32GB RAM) that hosts the Pattern

Orchestrator (PO) and the Message Broker (the latter based on Eclipse Mosquitto[8], leveraging the MQTT[9] lightweight, publish-subscribe network protocol). The PO is the component that is responsible for the process of translating the given orchestration descriptions into Drools facts. The Message Broker provides the means for the IoT sensors to be able to communicate their output to the Drools business rule management system. The second machine is a 64-bit ARMv8 Single Board Computer (more specifically, an Odroid C2, with Quad Core CPU, and 2GB RAM) that hosts the field Pattern Engine (PE) and acts as a gateway hypervisor. The large number of used sensors in the testing scenarios motivated the use of scripts to bootstrap the process; a python script was created for every orchestration with different number of sensors and was run on the backend server described above.

The orchestrations that are used for the evaluation are consisted by different number of temperature sensors, forming a series of nested merges. An orchestration of two sensors creates one merge. If three sensors are available, two of them are placed under a merge and the created merge along with the third sensor form a second merge. The maximum number of sensors that were used is 20.

The description of the IoT orchestrations is done through an editor in the IoT orchestration language, including all the involved components along with their desired properties and the orchestration-wide properties to be verified. Such an orchestration can be seen below:

1. **ORCH "5 temperature sensors"**
2. Iotsensor("IoTsensor1," "139.91.182.100," "9999," "00-80-e1-00-00-11"),
3. Iotsensor("IoTsensor2," "139.91.182.100," "9999," "00-80-e1-00-00-12"),
4. Iotsensor("IoTsensor3," "139.91.182.100," "9999," "00-80-e1-00-00-13"),
5. Iotsensor("IoTsensor4," "139.91.182.100," "9999," "00-80-e1-00-00-14"),
6. Iotsensor("IoTsensor5," "139.91.182.100," "9999," "00-80-e1-00-00-15"),
7. Iotgateway("Gateway," "6443," "139.91.58.100," "00-80-e1-00-00-32"),
8. Link("LS1M1," "IoTsensor1," "Merge1"),
9. Link("LS2M2," "IoTsensor2," "Merge2"),
10. Link("LS3M3," "IoTsensor3," "Merge3"),
11. Link("LS4M4," "IoTsensor4," "Merge4"),
12. Link("LS5M4," "IoTsensor5," "Merge4"),
13. Link("LM1G1," "Merge1," "Gateway"),
14. Link("LM2M1," "Merge2," "Merge1"),
15. Link("LM3M2", "Merge3", "Merge2"),
16. Link("LM4M3," "Merge4," "Merge3"),
17. Merge("Merge4," "IoTsensor4," "IoTsensor5," "Merge3," "LS4M4," "LS5M4"),
18. Merge("Merge3," "IoTsensor3," "Merge4," "Merge2," "LS3M3," "LM4M3"),
19. Merge("Merge2," "IoTsensor2," "Merge3," "Merge1," "LS2M2," "LM3M2"),
20. Merge("Merge1," "IoTsensor1," "Merge2," "Gateway," "LS1M1," "LM2M1"),
21. Property("Prop1," required, "Dependability," "0.0," datastate, Verification(monitoring, interface), "Merge1," false)

The testing scenario needs the involved sensors to communicate heartbeats and data (temperature) to the Drools business rule management system. This is done through publishing in appropriate topics of the Eclipse Mosquitto message broker.

The components of an orchestration (sensors, links, properties, etc.) are translated into facts and placed in the KB of Drools engine. The process of translated the given orchestration descriptions into Drools facts (performed by PO) is distinguished from the processed of inserting facts into the KB and performing the reasoning through rule triggering (performed by PE). The described orchestration is communicated to the PO, where a parser distinguishes the orchestration components and creates corresponding Java class objects. Said objects are sent to the PE to be included into the Drools memory as facts. Additionally, the produced from the sensors heartbeats and data (temperatures) are also sent to the PE since they are also considered Drools facts and are necessary for rule triggering.

Figure 3 depicts the reasoning time of both PO and PE and how it is affected by the number of sensors in the used orchestrations. In the case of PO, the reasoning time starts at 27 s when the incoming orchestration consists of 2 sensors and reaches 3 min and 7 s when the sensors participating in the orchestration are 20. Clearly, the more complex orchestrations are the more time consuming the reasoning of the Pattern Orchestrator is. The same pattern is observed for the PE. The numbers are similar, starting from 24 s when an orchestration of 2 sensors is coming as input, and reaching 4 min and 16 s when the sensors are 20.

The facts that represent orchestration elements trigger Drools rules that may create even more facts, which in their turn will make more rules to be triggered. As a result, more complex orchestrations lead to more facts, which in their turn lead to even more triggered rules, increasing the overall reasoning time.

Figure 4 shows that the memory footprint of the PE is also affected by the complexity of the given orchestrations as far as number of sensors is concerned. The numbers used for this graph were captured using top command of Unix, and more precisely, the %MEM column heading that stands for the share of physical memory used by a process. This does not include data that has been swapped to disk. The average of the values under the %MEM column heading of the PE, the rule engine that runs the Drools rules, is depicted in this figure, showing that the memory footprint increases, starting from 17.06% when the incoming orchestration consists of 2 sensors and reaching 18.23% when the most complicated orchestration of the evaluation is sent (20 sensors). Since the available RAM of the host is 2GB, 17.06% is translated into 349.38 MB and 18.23% corresponds to 373.35 MB, respectively. A different host with more than 2 GB RAM, which is not uncommon, will result in smaller percentage increase.

[8]Available online at: https://mosquitto.org/.
[9]Available online at: https://mqtt.org/.

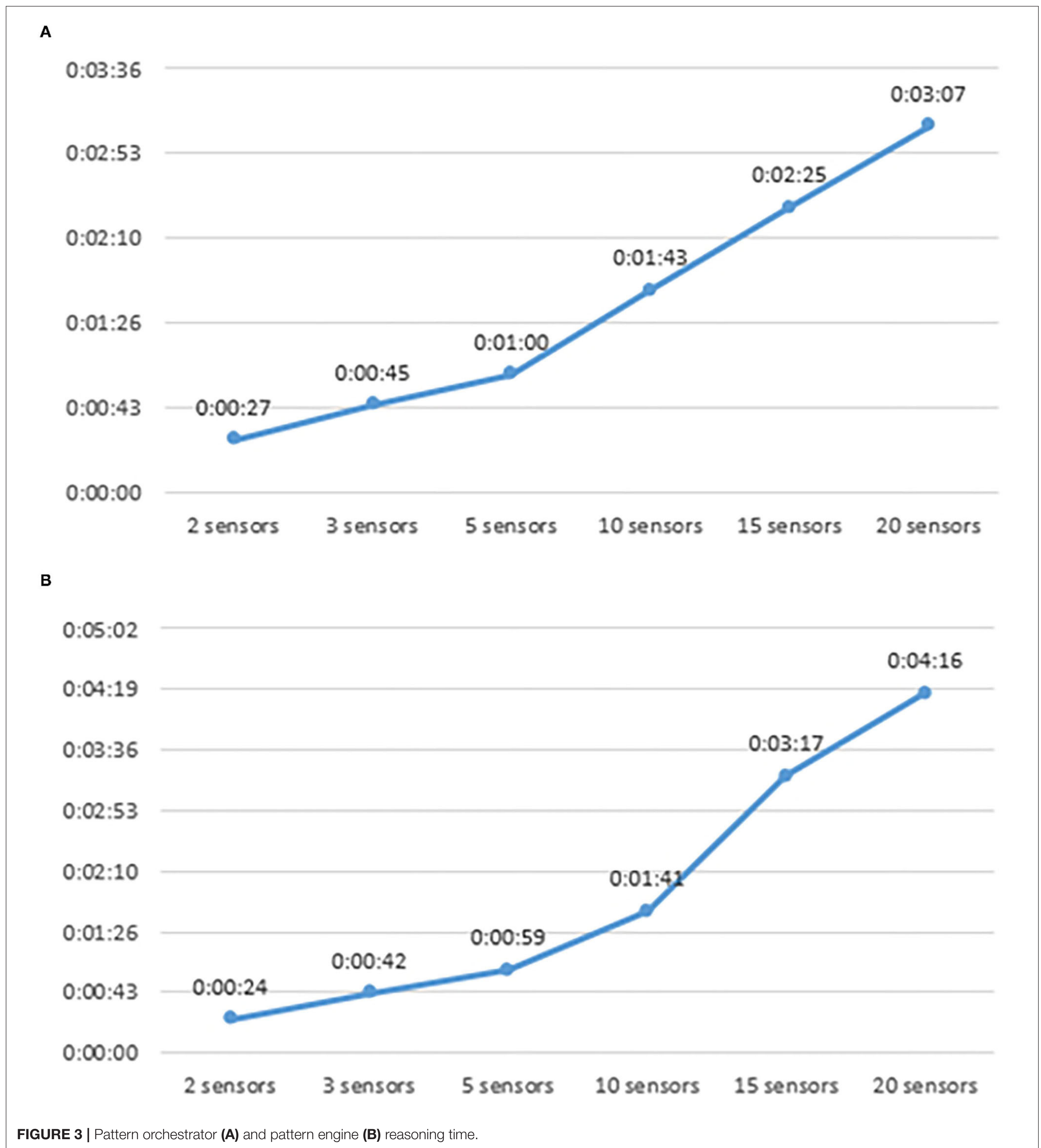

FIGURE 3 | Pattern orchestrator **(A)** and pattern engine **(B)** reasoning time.

DISCUSSION

The previous subsections presented the CIRCE approach for circular and trustworthy by-design IoT orchestrations. The presented pattern-driven framework facilitates the design-time specification of circularity and trustworthiness properties, through the same interface that the designer will use to specify the orchestration itself, while also supporting the runtime reasoning that these properties hold at runtime (within and across IoT deployment layers), and allowing to trigger any needed adaptations, to maintain and/or revert to these desired properties.

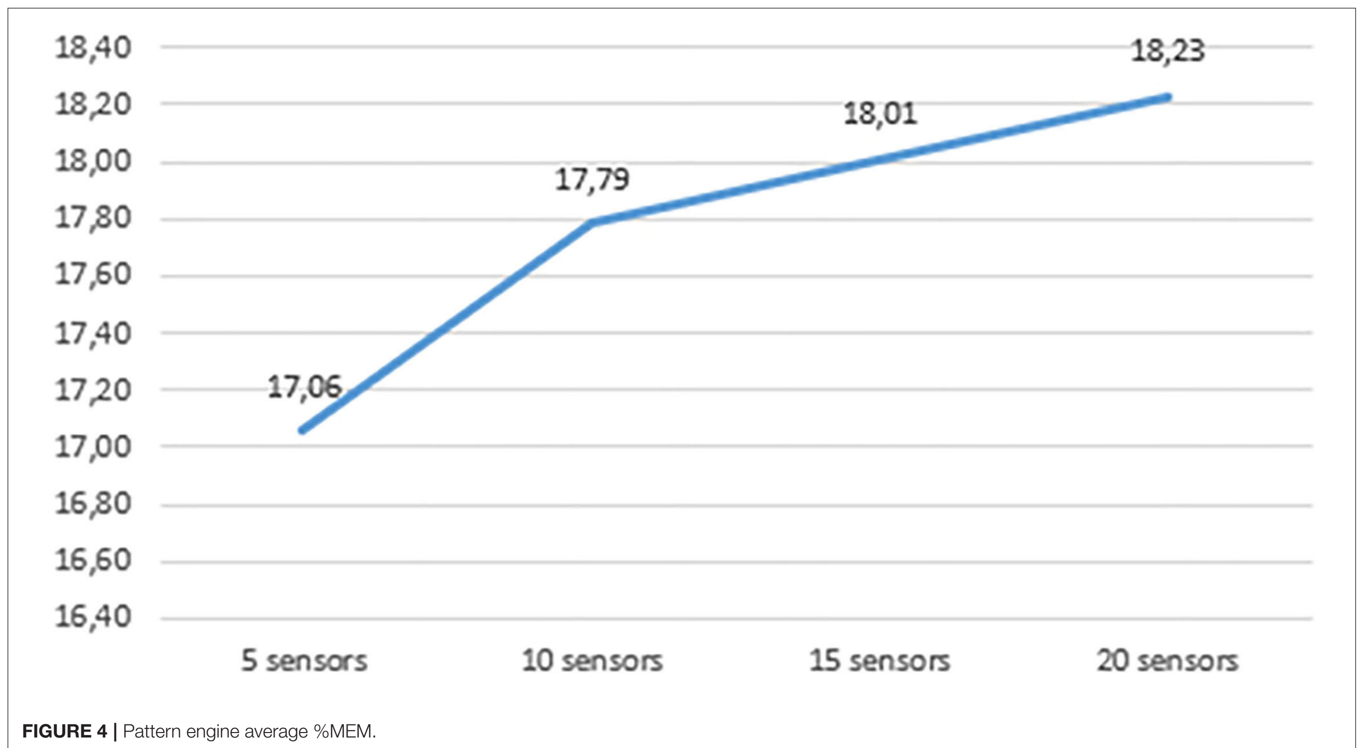

FIGURE 4 | Pattern engine average %MEM.

The evaluation results of the PoC implementation validate the scalability and general feasibility of the proposed approach. Nevertheless, as CIRCE is a complex framework, there are a number of challenges and risks that provide factors of uncertainty regarding the viability of a providing a practical solution that can be applied in real, complex real-world vertical applications.

In this context, a number of positive (i.e., risk mitigating) points can be highlighted as CIRCE framework is built on extensive expertise and mature core building blocks, including: (i) the Drools business rules and associated reasoning engines that have already been tested various applications domains and use cases; (ii) hands-on experience with integrating Drools with various user-friendly and open source tools that can be used for IoT Orchestration specification (e.g., Thuluva et al., 2017; Papoutsakis et al., 2020); (iii) ever-increasing visibility and interest on the benefits of the interplay between IoT and CE, with resources being committed to investigated this further (e.g., through involvement in several research projects that consider these aspects). The above provide a solid foundation for the further refinement and enhancement of CIRCE, with minimal risks from an implementation perspective.

Nevertheless, a number of open issues (and associated uncertainty factors) still remain that need to be addressed, before the full potential of CIRCE can be reached; these include: (a) application on heterogeneous vertical domains to further extend the System Model and the associated language to cover the intricacies of these domains, such as domain-specific devices (e.g., from self-driving tractors to manufacturing robotic arms and wearable medical devices); (b) definition of additional CSPDI & LCA patterns, both generic to cover additional sub properties (e.g., privacy patterns to allow verification of properties such as pseudonymity, undetectability, etc.) as well domain-specific, tailored to the requirements of different domains and end-users; (c) design and development of a pattern selection mechanism which, along with an associated pattern storage and indexing mechanism, will assist IoT orchestration designers in the selection of appropriate patterns (e.g., allowing the designer to select a higher level property, such as "Privacy," and then recommending or pre-selecting relevant lower-level properties, such as "Anonymity," that are appropriate for each activity comprising the specific orchestration); (d) validation of the proposed approach in an operational environment (especially considering the more novel Circularity properties, which have not been as extensively studied and tested as the CSPDI properties). The authors already investigate and intend to pursue all of the above points, which will be presented in future updates of CIRCE.

Concluding, in addition to the specifics of CIRCE, this work also intended to highlight the importance of the interplay between CE and technological developments in the IoT and 5G fronts. To reap the full benefits of this interplay, the co-design and co-development for Circularity and Trustworthiness properties within IoT building blocks is needed, facilitating the definition of Circular and Trustworthy by-design applications and services. CIRCE is a small step toward this direction, and the authors hope this approach and its prototypical implementation will inspire additional research in this relatively unexplored field.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

GS and SI: conceptualization, supervision, and funding acquisition. KF, GS, and SI: methodology and formal analysis. MP and EM: software. KF: validation. MP and KF: investigation, writing—original draft preparation, and visualization. MP, KF, and EM: data curation. GS, SI, and EM: writing—review and editing. All authors have read and agreed to the published version of the manuscript.

FUNDING

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 833828 (C4IIoT).

ACKNOWLEDGMENTS

The authors would like to thank all colleagues and collaborators involved in the CE-IoT, Ideal-Cities, and SEMIoTICS EU-funded research projects that paved the way for the creation of CIRCE.

REFERENCES

Ahituv, N., Lapid, Y., and Neumann, S. (1987). Processing encrypted data. *Commun. ACM* 30, 777–780. doi: 10.1145/30401.30404

Alexander, C. (1979). *The Timeless Way of Building*. Oxford: Oxford University Press.

Alrifai, M., Risse, T., and Nejdl, W. (2012). A hybrid approach for efficient Web service composition with end-to-end QoS constraints. *ACM Transact. Web* 6, 1–31. doi: 10.1145/2180861.2180864

Asghari, P., Rahmani, A. M., and Javadi, H. H. S. (2018). Service composition approaches in IoT: A systematic review. *J. Network Comp. Appl.* 120, 61–77. doi: 10.1016/j.jnca.2018.07.013

Askoxylakis, I. (2018). "A framework for pairing circular economy and the Internet of Things," in *2018 IEEE International Conference on Communications* (Kanas City, MO: IEEE). doi: 10.1109/ICC.2018.8422488

Botta, A., De Donato, W., Persico, V., and Pescapé, A. (2016). Integration of cloud computing and internet of things: a survey. *Futur. Gener. Comput. Syst.* 56, 684–700. doi: 10.1016/j.future.2015.09.021

Caiza, J. C., Martín, Y. S., Del Alamo, J. M., and Guamán, D. S. (2017). "Organizing design patterns for privacy: A taxonomy of types of relationships," in *ACM International Conference Proceeding Series; Association for Computing Machinery* (New York, NY). doi: 10.1145/3147704.3147739

CE-IoT (2020). *Deliverable D2.1, LCA and CSPDI Patterns*. Available online at: https://www.ce-iot.eu/wp-content/uploads/2020/06/CE-IoT-D2.1.pdf

Chung, E. S., Hong, J. I., James, L., Prabaker, M. K., Landay, J. A., and Liu, A. L. (2004). "Development and evaluation of emerging design patterns for ubiquitous computing," in *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques* (Cambridge, MA). doi: 10.1145/1013115.1013148

Del Borghi, A., Gallo, M., Strazza, C., Magrassi, F., and Castagna, M. (2014). Waste management in Smart Cities: the application of circular economy in Genoa (Italy). *Impr. Progetto Elect. J. Manage.* 4, 1–13.

Drozd, O. (2015). "Privacy pattern catalogue: a tool for integrating privacy principles of ISO/IEC 29100 into the software development process," in *IFIP International Summer School on Privacy and Identity Management* (Cham: Springer). doi: 10.1007/978-3-319-41763-9_9

Ellen MacArthur Foundation and the McKinsey Center for Business and Environment (2015). *Growth Within: A Circular Economy Vision for a Competitive Europe*. Available online at: https://www.mckinsey.com/business-functions/sustainability/our-insights/growth-within-a-circular-economy-vision-for-a-competitive-europe (accessed October 6, 2021).

Fawzy, S., Osman, A. I., Doran, J., and Rooney, D. W. (2020). Strategies for mitigation of climate change: a review. *Environ. Chem. Lett.* 2020, 1–26. doi: 10.1007/s10311-020-01059-w

Fysarakis, K., Papoutsakis, M., Petroulakis, N., and Spanoudakis, G. (2019). "Towards IoT orchestrations with security, privacy, dependability and interoperability guarantees," in *Proceedings of the 2019 IEEE Global Communications Conference (GLOBECOM 2019)*, (Waikoloa, HI). doi: 10.1109/GLOBECOM38437.2019.9013275

Hafiz, M. (2013). A pattern language for developing privacy enhancing technologies. *Software* 43, 769–787. doi: 10.1002/spe.1131

Hatzivasilis, G., Askoxylakis, I., Anicic, D., Broring, A., Kulkarni, V., Fysarakis, K., et al. (2018). "The interoperability of things: interoperable solutions as an enabler for IoT and Web 3.0," in *Proceedings of the 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)* (Barcelona). doi: 10.1109/CAMAD.2018.8514952

Hatzivasilis, G., Christodoulakis, N., Tzagkarakis, C., Ioannidis, S., Demetriou, G., Fysarakis, K., et al. (2019). "The CE-IoT framework for green ICT organizations: The interplay of CE-IoT as an enabler for green innovation and e-waste management in ICT," in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)* (IEEE). doi: 10.1109/DCOSS.2019.00088

Ideal-Cities (2020). *Deliverable D2.3, IDEAL-CITIES Patterns*. Available online at: https://www.ideal-cities.eu/wp-content/uploads/2020/05/IDEAL-CITIES_D2.3.pdf

ISO/IEC 29100:2011. (2011). *Information Technology—Security Techniques—Privacy Framework*. Available online at: https://www.iso.org/obp/ui/#iso:std:iso-iec:29100:ed-1:v1:en (accessed October 6, 2021).

Kalmykova, Y., Sadagopan, M., and Rosado, L. (2018). Circular economy –From review of theories and practices to development of implementation tools. *Resources* 135, 190–201. doi: 10.1016/j.resconrec.2017.10.034

Kristoffersen, E., Blomsma, F., Mikalef, P., and Li, J. (2020). The smart circular economy: A digital-enabled circular strategies framework for manufacturing companies. *J. Bus. Res.* 120, 241–261. doi: 10.1016/j.jbusres.2020.07.044

Kuhn, C., Beck, M., Schiffner, S., Jorswieck, E., and Strufe, T. (2019). On privacy notions in anonymous communication. *Proc. Priv. Enhancing Technol.* 105–125. doi: 10.2478/popets-2019-0022

Lahti, T., Wincent, J., and Parida, V. (2018). A definition and theoretical review of the circular economy, value creation, and sustainable business models: where are we now and where should research move in the future?. *Sustainability* 10:2799. doi: 10.3390/su10082799

Lecue, F., and Mehandjiev, N. (2010). Seeking quality of web service composition in a semantic dimension. *Transact. Knowledge Data Eng.* 23, 942–959. doi: 10.1109/TKDE.2010.237

Lee, I., and Lee, K. (2015). The Internet of Things (IoT): Applications, investments, and challenges for enterprises. *Bus. Horiz.* 58, 431–440. doi: 10.1016/j.bushor.2015.03.008

Lenhard, J., Fritsch, L., and Herold, S. (2017). "A literature study on privacy patterns research," in *Proceedings of the 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (Vienna). doi: 10.1109/SEAA.2017.28

Miaoudakis, A., Fysarakis, K., Petroulakis, N., Alexaki, S., Alexandris, G., Ioannidis, S., et al. (2020). "Pairing a circular economy and the 5G-enabled internet of things: Creating a class of ?looping smart assets?," in *IEEE*

Vehicular Technology Magazine (IEEE), 15, 20–31. doi: 10.1109/MVT.2020.2991788

Musti, K. S. (2020). "Circular economy in energizing smart cities," in *Handbook of Research on Entrepreneurship Development and Opportunities in Circular Economy* (IGI Global). doi: 10.4018/978-1-7998-5116-5.ch013

Papoutsakis, M., Fysarakis, K., Mixalodimitrakis, E., Lakka, E., Petroulakis, N., Spanoudakis, G., et al. (2020). "A pattern–driven adaptation in iot orchestrations to guarantee SPDI properties," in *International Workshop on Model-Driven Simulation and Training Environments for Cybersecurity* (Cham: Springer). doi: 10.1007/978-3-030-62433-0_9

Papoutsakis, M., Fysarakis, K., Spanoudakis, G., and Ioannidis, S. (2021b). "Defining IoT orchestrations with security and privacy by design: A gap analysis," in *IEEE Internet of Things Magazine* (IEEE), 4, 80–87. doi: 10.1109/IOTM.0001.2000162

Papoutsakis, M., Fysarakis, K., Spanoudakis, G., Ioannidis, S., and Koloutsou, K. (2021a). Towards a collection of security and privacy patterns. *MDPI Appl. Sci.* 11:1396. doi: 10.3390/app11041396

Petroulakis, N. E. (2018). Reactive security for SDN/NFV-enabled industrial networks leveraging service function chaining. *Transact. Emerg. Telecommun. Technol.* 29:e3269. doi: 10.1002/ett.3269

Pfitzmann, A., and Hansen, M. (2010). *A Terminology for Talking About Privacy by Data Minimization: Anonymity, Unlinkability, Undetectability, Unobservability, Pseudonymity, and Identity Management*. Tech. Univ. Dresden.

Razzaque, M. A., Milojevic-Jevric, M., Palade, A.,., and Cla, S. (2016). Middleware for internet of things: A survey. *IEEE Internet Things J.* 3, 70–95. doi: 10.1109/JIOT.2015.2498900

Schumacher, M. (2003). *Security Engineering with Patterns: Origins, Theoretical Models, and New Applications*. Berlin: Springer Science & Business Media.

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., and Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering*. Hoboken, NJ: John Wiley.

Seeger, J., Deshmukh, R. A., and Broring, A. (2018). Running distributed and dynamic IOT choreographies. *Global Internet Things Summit* 2018, 1–6. doi: 10.1109/GIOTS.2018.8534570

SEMIoTICS (2020). *Deliverable D5.9, Demonstration and Validation of IWPC - Energy Use Case (Cycle 2)*. Available online at: https://www.semiotics-project.eu/wp-content/uploads/2021/05/SEMIoTICS-D5.9_revised.pdf

Stallings, W., Brown, L., Bauer, M. D., and Bhattacharjee, A. K. (2012). *Computer Security: Principles and Practice*. Upper Saddle River, NJ: Pearson Education.

Steel, C., Nagappan, R., and Lai, R. (2005). *Core Security Patterns: Best Practices and Strategies for J2EE(TM), Web Services, and Identity Management*. Upper Saddle River, NJ: Prentice Hall PTR.

The World Bank (2019). *Solid Waste Management*. Available online at: http://www.worldbank.org/en/topic/urbandevelopment/brief/solid-waste-management (accessed October 6, 2021).

Thuluva, A. S., Bröring, A., Medagoda, G. P., Don, H., Anicic, D., and Seeger, J. (2017). "Recipes for IoT applications," in *Proceedings of the Seventh International Conference on the Internet of Things* (Linz: Association for Computing Machinery). doi: 10.1145/3131542.3131553

Conflict of Interest: KF was employed by Sphynx Analytics Ltd.

The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

The handling Editor declared a past co-authorship with several of the authors SI, GS, and KF.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

Copyright © 2022 Papoutsakis, Fysarakis, Michalodimitrakis, Spanoudakis and Ioannidis. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.