INVITED PAPER

Dependable Resource Coordination on the Edge at Runtime

This article introduces a methodology and a technical framework for engineering resource coordination for the edge-enabled Internet of Things (IoT).

By Christos Tsigkanos, Ilir Murturi, and Schahram Dustdar, *Fellow IEEE*

ABSTRACT | Software components within heterogeneous devices of the Internet of Things (IoT) systems use resources representing various computational capabilities, including sensing or actuation end points. However, components do not live in isolation and must be able to coordinate with others to fulfill their goals. Satisfaction of requirements—capturing their goals—must persist in environments that are changing, unpredictable, and potentially unknown at system design time. Edge computers placed near IoT devices can be leveraged for this sort of control—providing resource management for end devices within their operational context. We propose a methodology and technical framework for engineering resource coordination at runtime, tailored for the decentralized, pervasive systems of today. Our approach represents a paradigm shift in marrying distributed systems and formal aspects of software engineering. We adopt goal modeling to capture objectives within the system and use bounded model checking as the foundational technique to compute coordination plans that satisfy device goals. This occurs opportunistically at runtime without any knowledge about the operational status or presence of resources, but always in accordance with the edge's own goals. Our technical framework exhibits dependability guarantees regarding optimality and correctness of generated plans. We evaluate the resource coordination performance and its feasibility on low-powered ARM-based edge devices.

KEYWORDS | Dependable systems; edge computing; Internet of Things (IoT); model checking; software engineering

Manuscript received February 1, 2019; revised April 1, 2019; accepted May 8, 2019. Date of publication June 7, 2019; date of current version August 5, 2019. This work was supported in part by the TU Vienna Research Cluster SmartCT. *(Corresponding author: Christos Tsigkanos.)*

The authors are with the Institute of Information Systems Engineering, Distributed Systems Group, TU Wien, 1040 Vienna, Austria (e-mail: christos.tsigkanos@dsg.tuwien.ac.at).

Digital Object Identifier 10.1109/JPROC.2019.2917314

I. INTRODUCTION

Internet of Things (IoT) systems integrate heterogeneous devices, computing infrastructure, and cloud services with their ambient environments. New challenges and opportunities arise as rapidly growing cloud computing, mobile devices, sensors, and networks constitute larger ensembles of systems [1], [2]. A neighborhood, for example, may be saturated with hundreds of networked devices providing information to roaming humans or other devices, by combining information as they become available in the city's environment. Dynamic resource management [3] is essential to achieving such pervasive behavior—it enables devices and services making up the IoT to perceive available resources, configure them, and utilize them. Such resources may refer to computational, sensing, or other types of domain-specific resources that software-intensive devices may take advantage of to achieve their goals.

IoT applications differ in type and complexity, with multiple system components deployed in diverse domains and environments that are often not known beforehand. Moreover, software components within devices making up the system do not live in isolation and must be able to coordinate with others to fulfill their objectives as well as overall system-wide requirements [4]. Correct satisfaction of requirements must persist in environments that are changing, unpredictable, and potentially unknown at system design time [5], [6].

Resources within IoT applications may represent various computational capabilities, including sensing or actuation end points and storage or processing facilities. Often, those are architecturally abstracted as software services [7], referring to some functionality that different client IoT devices can reuse for different purposes. The concept of an IoT resource amounts to blurring the lines between

software services and sensor values or actuation end points.

We are not concerned with mechanisms of access, their interfaces or policies here, but with the fact that different interdependent resources may be required to fulfill some objective of a software component residing in a device. Dependences may be in the form of certain constraints—a resource to be operationalized may require the output of another, but its availability makes other resources additionally available. Dependences may be specified in an implementation- and language-agnostic manner and annotated over arbitrary resources that an application may use. We adopt an everything-as-a-service (XaaS) abstraction to uniformly represent physical things, hardware and software resources as microservices, irrespective of their specific nature [8]–[10].

Recent developments within distributed systems have led to the architectural placement of a computing entity closer to the network edge, close to IoT end devices, thus better satisfying system-wide goals, such as high availability, performance, or privacy [11]. Such edge entities may offer computation and control facilities to local devices [12]. Within a neighborhood for example, IoT devices may utilize resources of a local edge node benefiting from high connectivity to it as well as its awareness of other IoT devices in its scope. This allows an edge device to act as a mediator among devices, locally coordinating them in order to satisfy their resource needs. We build on the foundational edge concept where edge computers are placed near IoT devices, within their local administrative domain or wireless network. We further advocate decentralization, as the edge is a first-class entity in our approach, responsible for IoT devices within its scope but bearing no dependences for coordination to other edge nodes or the cloud.

We recognize that edge computing means different things to different people; we identify an edge node as a low-powered computer part of an IoT deployment. The edge node is in the scope of connected devices whose software stacks are limited. Although the method and framework proposed are hardware-architecture-free, we consider low-powered edge devices that are ARM-based and resource-constrained IoT devices such as microcontrollers populating the system as is the case in deployments of networked actuators and sensors, e.g., in smart cities. We treat communication and operational aspects as orthogonal to our approach; we are concerned with the core mechanisms of coordination at runtime.

To this end, we propose a methodology and technical framework for engineering resource coordination for the edge-enabled IoT. Our coordination approach targets decentralized edge systems, where IoT devices advertize and request resources at their local edge node. If an IoT device requests an edge node for a resource that cannot be trivially obtained from resources readily available, some combination of resources of other IoT devices must be derived. For example, computing a local weather forecast in a smart agriculture setting may require temperature readings from available sensors across a crop field. To solve this, coordination on the part of the edge node computes a plan, in which the requesting device can use to fulfill its objective, in accordance with the edge's goals. Thus, resources are coordinated regionally within the local IoT scope of the edge node. Overall, system-wide goals may further be affected by coordination occurring in a scope. Our concrete contributions are as follows.

1) We provide a methodology where semantic annotations are specified at design time to arbitrary resources within an IoT system. Those record what a resource requires to be operational and what effects its potential operationalization has on other resources or context values. Subsequently, the objectives of entities within the IoT system are specified—from a requirements' engineering perspective, our approach is goal-driven since we use edge and device goals to drive coordination of resources at runtime.
2) When the IoT system is operational, a boolean satisfiability problem (SAT)/satisfiability modulo theories (SMT) [13] solver situated on a low-powered edge device leverages bounded model checking techniques [14] at runtime to fulfill the objectives of local IoT devices by coordinating available resources in its scope based on the currently active context.

We instrument coordination as a form of service composition [15], but tailored for the edge-enabled IoT. While building upon the significant state of the art of traditional service composition, our resource coordination technique differs for three key reasons.

1) We consider elementary IoT resources as microservices—instead of using a service description language [16]–[18], we adopt a lightweight approach suitable for microservices inherent in modern IoT architectures and applications.
2) We allow quantifiers and integer linear arithmetic for specification due to the IoT domain.
3) We target low-powered ARM-based edge computers for deployment.

Our approach represents a paradigm shift in marrying distributed systems and formal aspects of software engineering. Specifically, we adopt goal modeling to model objectives within IoT. We use bounded model checking [19] as the foundational technique to compute coordination plans that satisfy device, edge, and system goals. This occurs opportunistically at runtime, without any knowledge about the operational status of the system or which resources are present at the system's design time. The coordination facilities we provide are dependable because if there is a solution to a resource coordination problem, the technique we utilize will provide a plan for it and the plan will be optimal. This is in contrast to other approaches utilizing other techniques such as based on AI [20]–[22]. We acknowledge that the

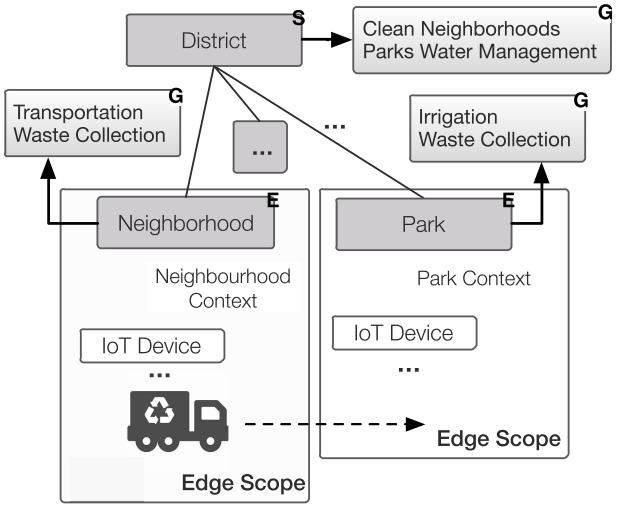

Fig. 1. *IoT resources within a smart city.*

technique we adopt is computationally expensive, but it offers dependability guarantees. Thus, our evaluation targets resource coordination performance and its feasibility on low-powered ARM-based edge devices.

The rest of this paper is structured as follows. After a motivating example used throughout this paper in Section II, Section III gives an overview of our approach within edge computing. Section IV describes key modeling and methodological aspects, goal and modeling of which are expanded in Section V. Subsequently, Section VI illustrates bounded model checking for resource coordination. Section VII provides an assessment of the feasibility and performance of the proposed approach. A related work is considered in Section VIII, and finally, Section IX concludes this paper.

II. RUNNING EXAMPLE

As a simple scenario serving as a running example of an IoT system throughout this paper, consider a modern smart city containing various neighborhoods and parks where various devices are embedded providing smart functionalities. Naturally, waste bins are located in neighborhoods as well as parks, and traffic lights may be deployed to facilitate municipal vehicles; ambulances or recycling trucks should be presented with green traffic lights when applicable. Moreover, irrigation facilities situated in parks should automatically be operational when the detected soil moisture is below a threshold of 20%. However, this should not occur when the park is crowded with visitors. Besides ensuring municipal vehicle green lights, the city management imposes other constraints as well, regarding the overall system's operation. In order to minimize citizen disruption, irrigation and waste collection in parks must not occur simultaneously. Moreover, irrigation should not be active in more than one park at a time to conserve water.

Notice that the smart city presented is an instance of the IoT; several sensing or actuating devices are needed to realize it. Devices or certain scopes in the city (e.g., neighborhoods or parks) have various goals, which, when the system is in operation, may conflict (e.g., moisture may drop while a recycling truck arrives in a park). Moreover, the particular conditions and configurations of IoT devices are unknown at design time; we require no knowledge of recycling trucks, presence or not of irrigation in parks, for example. It is then evident that coordination among IoT devices is required to fulfill their various goals. Note how city goals consist of system-wide goals that may be affected by the IoT devices and edge nodes within it.

III. COORDINATION ON THE EDGE

IoT applications can be of various types, software stacks, and complexities, with multiple system components deployed in diverse domains and contexts. Those, however, do not live in isolation and must be able to coordinate to fulfill application or end-user requirements [4]. A software component hosted on some device, for instance, may require reading from a sensing end point in order to perform some computation and fulfill its objective. This problem is exacerbated within IoT deployments, as applications need to operate on diverse infrastructures and integrate heterogeneous components from various providers in a long-running system, with possibly conflicting goals between the components.

A. IoT Resources, Services, and Goals

IoT software components provide data, sensing and actuation, as well as computational resources to other software components, which can be abstractly represented with the concept of an IoT resource [10]. Within an IoT system, components can have different software stacks but still interact—this is widely achieved by software services, the architectural abstraction permeating many systems today [23]. A system's development is then based on writing custom business logic that utilizes services. As IoT components are resource-constrained, services often take the form of loosely coupled microservices, communicating with lightweight methods. Examples of this are typically found as sensing or actuation end points—a temperature sensor responds with a temperature value when invoked for instance or a smart door unlocks a door when the security system requires it to. Such functionalities may be abstracted as resource microservices that the software-enabled devices make available over their local scope such as a wireless network.

Resources that an IoT device may need from others need to be appropriately composed and communicated to the device in order for it to fulfill its objective. For example, computing a local weather forecast (i.e., an objective) in a smart agriculture setting may require temperature readings (i.e., resources) from available devices across a crop field. In the general sense, resources available within an IoT scope—some local context—need to be

coordinated, with the IoT device's goal in mind. This coordination essentially amounts to planning as understood within self-adaptive systems: actively setting in motion configuration changes to satisfy certain objectives, in this case the goal of an IoT device that depends on other IoT devices' resources in its local scope. Satisfying an IoT component's goal, however, is challenging, as devices are deployed in changing and unpredictable (i.e., at design time) environments. Assumptions made at system design time about the availability or location of resources that a device needs may be violated. Thus, facilities providing control and coordination must be performed at runtime and based on the current environmental configuration, by ensuring that the IoT system can autonomously react to the changes in different contexts in a dependable manner.

B. Coordinating Resources on the Edge

Centralizing computation of coordination—typically in the cloud and evident in today's IoT-cloud architectures—is one solution but requires cloud control structures to be always available and within low latency. However, novel functional and nonfunctional requirements that have arisen in IoT systems dictate computation and control to be situated locally near devices [2]. Centralized coordination on the cloud is naturally possible. However, the cloud (as a central point of failure) may not be available, it is found within high latency from local devices and would generate impractical, unnecessary network overhead, as every device would require coordination functionality to be communicated to the cloud and back for every resource request. We advocate that since the edge computing entity is closer to end devices (and IoT application users), there is an opportunity for situating coordination there—something realized by empowering an edge computer to actively coordinate resources of IoT devices within its scope. We note that this fits the domain particularly well; IoT devices are found within a local scope, such as a local wireless network or a deployment within a limited geographical region (e.g., a city neighborhood). As such, placing an edge computing entity close to a set of locally scoped devices providing coordination facilities is highly feasible.

Distributed systems' mechanisms relevant to process coordination and control, such as service engineering and resource management, can be adopted to identify and discover IoT resources. Methods developed within formal aspects of software engineering, such as requirements reasoning, model-driven planning, and self-adaptive systems, are then adopted in our approach to enable coordination of available resources at runtime. Models kept at runtime facilitate coordination and the determination of how control actions can satisfy goals within the system. Regarding architectural deployment, the edge is a first-class entity in our approach, acting as a manifestation of a control agent responsible for receiving IoT device resource requests, observing contextual information, and inducing appropriate actions to satisfy them.

C. Instrumenting Coordination

Fig. 2 shows a bird's-eye view of our approach to coordination at runtime on the edge. The capabilities offered by IoT devices' are abstracted as resources (i.e., microservices) and made available through the network. In our approach, those are specified at design time for each participating software-enabled device, together with possible goals that the device may seek to achieve (1). The resource configuration of the system as well as the environment that the system may be found when operational is unknown at design time. At runtime, IoT devices are found within some local scope and may interact with others to utilize their resources. However, device goals may have interdependent requirements on other resources, so coordination is required. An edge node situated close to the IoT end devices and managing the local IoT scope is responsible for coordinating available resources at runtime (2). Participating devices then interact according to generated coordination plans to fulfill their goals (3).

Resource coordination occurs opportunistically at runtime, for which we propose a technique based on bounded model checking. This offers the guarantees of correctness and optimality of the generated plan that satisfies a requesting IoT device's goal. The resource coordination we propose extends traditional service composition [24] and brings it into the IoT context: 1) we adopt a lightweight approach suitable for resource-constrained IoT microservices; 2) we allow quantifiers and integer linear arithmetic for specification; and 3) we target low-powered ARM-based edge computers for deployment. To completely realize the edge-based coordination facilities advocated from a systems' perspective, communication and operational aspects must be treated. This includes: 1) abstraction of resources from a programmatic perspective (e.g., how resources are annotated at development time); 2) how devices and edge nodes communicate; and 3) how the system copes with operational real-time constraints, since those can vary per deployment. For the latter, responses to resource requests from IoT devices must occur in a timely manner, as the environment changes rapidly (e.g., the recycling truck arrives at the traffic light, see Fig. 1).

IV. DOMAIN AND METHODOLOGY

In this section, we provide the basic abstractions and methodological principles necessary to instrument coordination within an IoT domain we are situated in. For formalization purposes, we assume a global set of names or key-value pairs Π that appear throughout the system.[1] We begin by outlining key elements and assumptions of our approach, upon which we define a methodology that the system designer follows to instrument resource coordination at the edge.

[1]Without loss of generality, we take Π to comprise atomic propositions, essentially mapping identifiers and their values to true statements.

A. IoT Resource

Architecturally, IoT devices are software components deployed in different environments, each containing resources. Generally, we assume that an IoT system is architecturally composed of processes that are microservices [8]–[10]. Such IoT microservices when invoked yield resources; however, successful invocation entails meeting the requirements of a microservice, which may depend on others. We will refer to microservices and the resources that they yield interchangeably. This may occur for several sequences of resource invocations, thus motivating the need for coordination; knowledge of which resources are needed to operationalize a resource that an IoT device requires entails coordination. IoT resources within our approach are implementation- and language-agnostic; what we require is the modeling of their preconditions (i.e., what they require to be activated) and their postconditions (i.e., how their successful activation changes some context).

For our example, we assume that three exemplar resources are present in the IoT system of the city, highlighting different modeling aspects. A recycling truck has a recycling_truck resource, which empties waste cans in a neighborhood where the recycling truck is present. A smart traffic_light ensures that municipal vehicles—such as the recycling truck—are met with green lights, and an irrigation actuator in a park is responsible for watering it when required. In-depth treatment of resources will be described in Section V.

B. Runtime Edge Context

In edge computing architectures, IoT end devices interact with their environment, where the edge device is by definition located within the local domain of certain IoT devices—one can take that as the devices being in the logical scope of the local edge node. The status of various devices, resources, as well as environmental information observed during system operation and the edge node is aware of is referred to as the runtime edge context. Edge context is assumed to be local to some edge node (see Fig. 2). We identify as $C \subseteq \Pi$ the runtime edge context, comprising of a set of key-value pairs within an edge scope. A valuation of C refers to a specific moment in time—key-value pairs reflect the runtime physical or logical environment. While keys are unique identifiers, the domain of their values can range. Specifically, we allow booleans, a domain of a finite set, or arbitrary integers. Valuation may change because of monitored information (i.e., resulting in a change to a sensor value) or due to exogenous to the system stimuli. However, it may also change due to resources of IoT devices: 1) if a resource is made available to others, this is reflected in C, and 2) if it is operationalized, it may change the values it its context. We assume appropriate instrumentation for the correct accounting of the various values within the edge context.

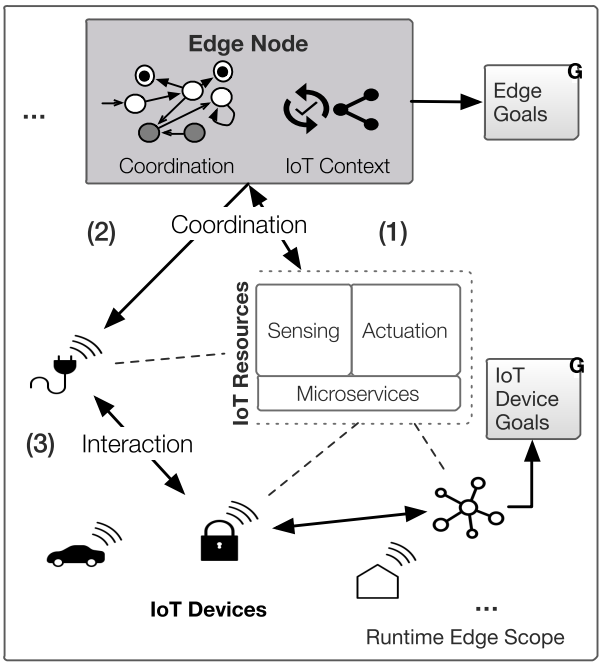

Fig. 2. ***Runtime resource coordination on the edge.***

For our example, neighborhoods and parks are edge scopes, in each of which an edge node is placed, accounting for the current context and IoT devices that may be nearby and, as we will observe later, coordinate their resources. We assume that waste_cans and traffic_light are two identifiers within the edge context of a neighborhood, values of which are populated by sensors in the waste cans and a traffic light IoT device, respectively [see (1)]. In the park, we assume that a sensor detects soil_moisture and the other senses if the park is crowded [see (2)]. Within a neighborhood context, we can observe different domains for values of its identifiers: while waste_cans can be true or false, traffic_light can be either green or red. Differently, soil moisture in a park can be some integer value

$$C_{\text{neighd}} = \{\text{waste\_cans} : \text{bool}, \text{traffic\_light} : \{\text{green}|\text{red}\}\} \quad (1)$$

$$C_{\text{park}} = \{\text{crowded} : \text{bool}, \text{soil\_moisture} : \text{int}\}. \quad (2)$$

C. IoT/Edge Goal

An objective that an IoT device or edge node seeks to achieve is referred to as a goal—satisfaction of a goal depends on available resources at its local environment. Goals capture at different levels of abstraction, the various objectives' entities within the IoT system under consideration should achieve, or constraints of various context values within their control. A goal for an IoT or edge device is a logical formula over the set Π.

Back to our running example (shaded boxes within Fig. 1), parks should be watered, but this should not occur simultaneously with waste collection—this

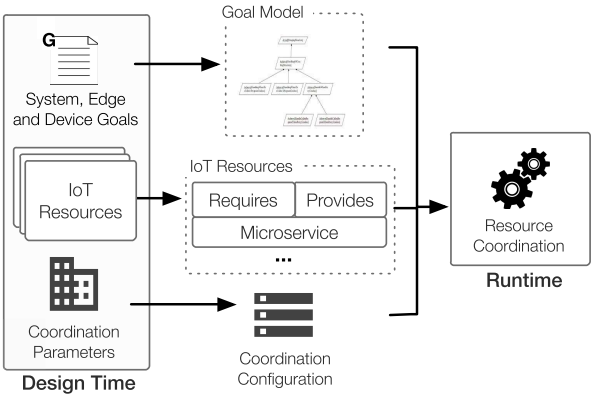

Fig. 3. ***Engineering coordination: design time methodology.***

entails an objective of the edge node placed in the park. Similarly, a neighborhood edge node should ensure that if municipal vehicles are present, green traffic lights allow them to pass. The overall city or district containing parks and neighborhoods imposes constraints, such as water management. Such goals will be modeled precisely in Section V.

D. Design Time Methodology

Our approach entails engineering resource coordination tailored to IoT systems, and it methodologically spans both design time and runtime. Illustrated in Fig. 3 by leveraging design-time specifications, coordination is enabled at runtime through the following steps.

1) *Modeling IoT Resources:* Language-agnostic, semantic annotations to arbitrary IoT resources of participating devices are specified. Such annotations record what a resource requires to be operational and what effects its invocation has on other resources or context variables. This step will be described in Section V-A.
2) *Modeling Device and Edge Goals:* Objectives of the various entities active within the system are captured in a goal model, which facilitates goal refinement, resolution of conflicts, and goal interdependences. This step will be described in Section V-B.
3) *Specification of Runtime Coordination Parameters:* The technique employed at runtime to satisfy resource requests from IoT devices is deployed on a resource-constrained edge device. As such, coordination parameters regarding performance aspects are specified depending on some particular deployment setting. The coordination technique is described in Section VI, while useful insights about parameterization will be illustrated in Section VII.

A further necessary step—out of the scope of this paper—are application-specific as well as architectural deployment aspects. For a complete instrumentation of coordination, edge nodes must be deployed and be responsible for certain scopes; communication and network management must be handled as well. We consider such aspects as orthogonal to our approach and assume that they are in place.

V. RESOURCES AND GOALS WITHIN IoT

Resources in an IoT context may be arbitrary, provided by heterogeneous software components deployed on devices from various vendors and architectural stacks. Within the overall IoT collective, devices and edge nodes alike may have objectives that they seek to achieve. In this section, we first describe how IoT resources can be generally represented. This includes particularly what they require to be operationalized and what their invocation entails for others. Second, we adopt requirements' engineering methods to capture objectives throughout the system by goal modeling.

A. Modeling Resources Within IoT

To model resources within an IoT system, we advocate the principle of procedural abstraction: capturing the knowledge of IoT process internals may be impractical, but considering the requirements and effects of IoT processes is feasible. For example, one does not need to know how a sensor array calculates mean temperature based on a spatial dispersion of IoT sensors, but only that by invoking some software service, the current average temperature is obtained. To this end, as noted in Section IV, we assume that a software-intensive IoT system is architecturally composed of processes that are microservices. Such IoT microservices, once invoked, yield resources; however, successful invocation entails meeting the requirements of a microservice, which in turn may depend on others. This is where the use of preconditions and postconditions is beneficial, which we informally refer to as what an IoT microservice requires and what it provides. Preconditions and postconditions are first-order logical formulas as parameters, with propositions in set Π. Quantifiers (over finite sets) and integer linear arithmetic may be used for specification.

1) *Requires* is a precondition directive that outlines what conditions should be true in a given context for a resource to become operational, essentially its requirements.
2) *Provides* refers to a postcondition directive that outlines what conditions are true as a result of an operationalization of a resource.

More formally, a resource is a tuple $\lambda = \langle \mathsf{R}_\lambda, \mathsf{P}_\lambda \rangle$ where R_λ and P_λ are the first-order formulas of input and output parameters, respectively. Parameters themselves are sourced from the global set of propositions Π. Without loss of generality, those are assumed to be key-value pairs. We slightly abuse notation and refer to $\mathsf{parameter}$ if $\mathsf{parameter} = \top$. Given the above, when a resource λ is invoked with input R_λ, λ returns output $\mathsf{p} \in \mathsf{P}_\lambda$. For every resource, *Requires* and *Provides* directives

are specified at the system's design time. For presentation purposes, we will write $[\mathsf{R}_\lambda]\ \lambda\ [\mathsf{P}_\lambda]$ and $\lambda = \langle \mathsf{R}_\lambda, \mathsf{P}_\lambda \rangle$ interchangeably

$$[\mathsf{municipal\_vehicle}]\ \mathsf{allow\_vehicle}\ [\mathsf{traffic\_light} = \mathsf{green}]. \quad (3)$$

Given the above, we can model the resources of our smart city example. Recall that traffic lights deployed facilitate municipal vehicles such as ambulances or recycling trucks so that they are presented with green traffic lights. This traffic light functionality can be represented as a resource (i.e., that a traffic light IoT device offers), setting the traffic light to green when applicable. Formula (3) intuitively states that when a municipal_vehicle context value is true, the light turns green. The context value within a neighborhood ($\mathsf{C}_{\mathsf{neighd}}$) is assumed to be set by, e.g., a truck when it is within the local scope of the traffic light. The functionality of a recycling truck (i.e., as an IoT device) can be similarly represented in the following:

$$\begin{bmatrix} \mathsf{traffic\_light} = \mathsf{green} \\ \wedge\ \mathsf{waste\_cans} = \mathsf{full} \end{bmatrix} \mathsf{recycling\_truck}[\mathsf{waste\_cans} = \mathsf{empty}]. \quad (4)$$

Quantitative values can also be captured in resource parameters—for this purpose, we support integer linear arithmetic. For example, the irrigation resource present in the park (i.e., due to an IoT device responsible for irrigation) should be activated when detected soil moisture is below a certain threshold and when the park is not crowded. The irrigation resource can then be modeled as in

$$\begin{bmatrix} \neg\ \mathsf{crowded} \wedge \\ \mathsf{soil\_moisture} < 20 \end{bmatrix} \mathsf{irrigation}\ [\mathsf{soil\_moisture} = 20]. \quad (5)$$

When some context value crowded is false and some other soil_moisture is less than 20, irrigation—if activated—will result in setting the latter to 20; those refer to $\mathsf{C}_{\mathsf{park}}$.

In general, resource models in IoT [25] are widely established in the literature and can be utilized to model resources as the formulas' tuples $\langle R_\lambda, P_\lambda \rangle$ we advocate, since the model is quite generic. Within parameters, propositions of $\langle R_\lambda, P_\lambda \rangle$ formulas can include: 1) location, describing the logical-physical domain where a resource resides; 2) administrative domain, describing a repository permitting authentication or authorization; 3) type, characterizing a resource instance as a sensor, actuator, or a logical entity; or 4) capability, providing special abilities that a resource enjoys, based on some domain ontology. All of those, including quantitative cases, can be encoded as $\langle R_\lambda, P_\lambda \rangle$ parameters and exposed in the namespace of an edge context using the method previously described.

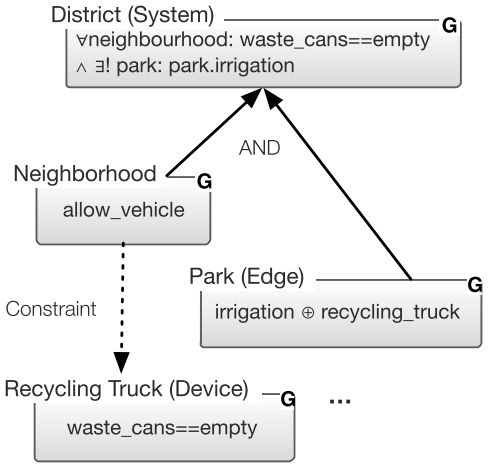

Fig. 4. *Goal model capturing objectives throughout the system.*

B. Modeling Goals Within IoT

Recall that both IoT devices and edge nodes may have goals; in our context, goals are objectives within the system that various entities seek to achieve. The role of the edge is to facilitate goal achievement for devices within its scope. Depending on the general state of the system, overall system-wide goals may be affected in turn.

Goal-oriented modeling, widely used in requirements engineering [26]–[29], is a technique that can capture, clarify, and enable an analysis of system requirements. The structured form of a goal model allows the refinement of system goals to subgoals, prioritization of requirements, as well as resolution of inconsistencies that may be due to conflicting stakeholder viewpoints. Our intuition to model the goals in the IoT-edge context is that edge nodes in IoT systems are often arranged in a hierarchy, where the cloud is the global or root entity representing the whole system, and edge computers are found within that hierarchy with IoT devices as end nodes. Notice that this structure is reflected in our running example, where a district may contain multiple neighborhoods and parks each having a logical edge node. IoT devices are within the scope of an edge entity leading to a tree arrangement where IoT devices are leaves.

In our approach, we adopt a form of discrete goal modeling to capture the objectives of devices, edge nodes, as well as their relationships.[2] As shown in the goal model of Fig. 4 which encodes system concerns of the running example, each goal can be refined into subgoals through an and–or decomposition. At the leaf level of the goal model reside IoT device goals—each leaf describes an objective of an IoT device. As with the specification of preconditions and postconditions, a goal for an edge device is an arbitrary first-order logical formula E over the set of global set of names Π; for an IoT device, we denote its goal as

[2]Note that weights can be assigned if quantitative aspects are desired, leading to a weighted goal model [30] and further constraints.

G. Quantifiers (over finite sets) and integer linear arithmetic may be used for specification.

Goal modeling is used to: 1) provide an indication of satisfaction of the various system objectives; 2) provide a structured way of managing relationships within the edge-intensive system; and 3) coordinate appropriate devices within edge scopes. The satisfaction status of each (sub)goal in the goal model (e.g., in Fig. 4) can be determined by observable runtime information or by active operations of IoT devices. The current context status on every edge node captures this. For example, the status of waste cans in a neighborhood can either be monitored (e.g., by sensors in the waste cans) or set to "empty" by a recycling truck. The goal model structure intuitively shows how other subgoals are affected—observe that if a recycling truck empties the waste cans in a neighborhood, its respective goal will be affected. Values of subgoals are propagated upward the goal model (i.e., over a system goal structure), affecting the satisfaction of parent goals. Assuming a variable with values of a known finite set of neighborhoods in a district, the system-wide goal states that for every neighborhood, the waste cans should be empty and that there is exactly one park such that the irrigation is true. Similarly, a park edge subgoal captures the fact that irrigation should not be true at the same time where a recycling truck is present in its scope.

Edge goals govern how they coordinate resources for requesting IoT devices—they constrain how objectives of IoT devices are achieved. Since the runtime edge context is unknown and coordination occurs at runtime, goal satisfaction happens opportunistically; edge nodes' and devices' goals may be satisfied depending on the presence of other devices and runtime context values. This, in turn, may affect other subgoals of the system—for example, if the waste cans are emptied in a neighborhood and this happens for every neighborhood in a city, the district's system-wide goal may be satisfied (the first clause in the conjunction in Fig. 4). Goal relationships between edge nodes are not accounted for coordination as this would impair system performance and incur centralization—each edge in a decentralized manner imposes its own goals within its scope, but their resulting satisfaction is propagated to parent system goals as subgoals. Thus, monitoring of system-wide goals is supported. Note that imposing system-wide goals to edge nodes would imply that each edge is constrained by what occurs within another edge's scope, something that would require a centralized (perhaps priority-based) strategy—we identify this as a promising avenue of future work. Overall, the specification of a goal model is left to the system designer, which may specify arbitrary goals for entities in the system.

VI. DEPENDABLE RESOURCE MATCHMAKING

As we observed, the edge as the coordinator within its active runtime context receives a request from a device seeking to achieve some goal which depends on other IoT resources or context values. Coordination then amounts to figuring out how to combine available resources or context values to produce a plan, which is then be returned to the device. We call this process resource matchmaking, as it entails making a match between the requesting device and other devices, such that their resource combination can achieve a goal. Matchmaking as described is a complex problem as it amounts to NP-completeness; in this section, we present the technique we utilize, which results in dependability guarantees; solutions are provided always correctly and optimally (if they exist).

To tackle resource matchmaking, after first formally defining the problem, we demonstrate how it can be mapped to a state-transition structure capturing the evolution of resources in the system. Subsequently, we reduce the matchmaking problem to reachability within this state-transition structure. Finally, we provide a conjunctive normal form (CNF) encoding of the problem that is suitable as an input to a solver, upon which bounded model checking [14], [19], [31] is used to solve it, yielding a correct and optimal solution.

A. Resource Matchmaking Problem

Recall that a resource is a tuple $\lambda = \langle \mathsf{R}, \mathsf{P} \rangle$, where R and P are first-order formulas of input and output parameters, respectively. We assume that when a resource λ is invoked with the input formula R, λ returns output P (i.e., resource microservices work correctly). To decide an invocation relationship from resource $\lambda_1 = (\mathsf{R}_1, \mathsf{P}_1)$ to $\lambda_2 = (\mathsf{R}_2, \mathsf{P}_2)$, it is necessary to compare the outputs P_1 of λ_1 with inputs R_2 of λ_2. To establish a relationship, the requirements R_2 of λ_2 must be met. Generally, given a set of available resources and a request resource λ_{req}, we seek to find a resource λ such that $\mathsf{R}_{\lambda_{\text{req}}} \subseteq \mathsf{P}_{\lambda}$. However, there might be the case that there is no single resource satisfying the requirement of the requesting resource. In that case, we seek to find a sequence $\lambda_1 \cdots \lambda_k$ of resources where, in each step, invocation of a resource λ_i occurs and the desired objective is eventually achieved. As there can be many such sequences, the optimal solution for the resource matchmaking problem is to find one with the minimum value for k.

B. Resource Evolution and Device Goal Reachability

To enable automated reasoning, we represent the evolution of resources in a state-transition system generally known as a (doubly) labeled transition system (dLTS) [32], which is a tuple $\mathcal{K} = (\mathcal{S}, \Pi, \Lambda, \mathcal{L}, \mathcal{A}, \mathcal{I}, \mathcal{G})$, where Π is the global, finite set of atomic propositions, $\mathcal{S}$ is a set of states, $\mathcal{L} : \mathcal{S} \rightarrow 2^{\Pi}$ is a function that labels each state with the set of propositions Π that are true in that state, Λ is a set of transition labels capturing resources, $\mathcal{A} \subseteq \mathcal{S} \times \Lambda \times \mathcal{S}$ is a 3-adic accessibility relation (if $p, q \in \mathcal{S}$

and $\alpha \in \Lambda$, then $(p, \alpha, q) \in \mathcal{A}$ is written as $p \xrightarrow{\alpha} q$), and $\mathcal{I} \in \mathcal{S}$ is an initial state and $\mathcal{G} \in \mathcal{S}$ is a device goal state.

States of $\mathcal{K}$ capture values (or parameter instantiations), while transitions record how those can change by moving from one state to its successors by operationalizing resources. Each state declaratively represents an instantiation of resources and context values (i.e., of Π) at some moment of time. The accessibility relation $\mathcal{A}$ between the states shows how parameters' instantiations and context values change moving from a state s to another s'; it corresponds to the transitions of $\mathcal{K}$. Intuitively, starting from an initial state of the system representing an initial configuration, application of resources λ_i generates states according to their R_{λ_i} and P_{λ_i} and context values.

Given an incoming request for G from a device, the initial state $\mathcal{I} \in \mathcal{S}$ captures the context values at the edge node at the time of the resource request (i.e., at runtime). The goal state $\mathcal{G}$ captures some configuration where G holds. Solving the matchmaking problem as presented amounts to reachability [33] of $\mathcal{G}$ within dLTS $\mathcal{K}$ as illustrated in the following.

C. Resource Matchmaking With Bounded Model Checking

As we observed, given a request for a goal G from a device, the desired outcome for the matchmaking problem is to find a sequence of resource applications which, starting from an initial state, bring the system to a state where G is fulfilled. In the following, we show how this reachability problem [33] can be solved with bounded model checking [19], [31] through an encoding to a CNF formula.

To formalize the reachability problem, it is first necessary to introduce the following definitions. Given that $s_i \in \mathcal{S}, 0 \leq i \leq n$, and $\alpha_i \in \Lambda$, a finite computation is defined as a finite composition of transitions

$$s_0 \xrightarrow{\alpha_1 \cdot \ldots \cdot \alpha_n} s_n =_{def} s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \; s_{n-1} \xrightarrow{\alpha_n} s_n.$$

The concatenation $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_n$ of labels (representing resource invocations) is called a trace originating from s_0. The sequence of states $s_1 \cdot \ldots \cdot s_{n-1}$ is called the sequence of traversed states. State s_0 is the originating state of the sequence and state s_n is the end state. Reachability entails the existence of a computation $\mathcal{I} \cdot s_1 \cdot \ldots \cdot s_{n-1} \cdot \mathcal{G}$. Each state s_i along the computation captures available values (or resource parameter instantiations) in time instant i. The desired outcome is the respective trace; if the requesting IoT device invokes the resources indicated by the trace in series, it can reach $\mathcal{G}$, where its goal G is fulfilled.

Recall that instantiated parameters and context values as propositions that live on states $\mathcal{S}$ are drawn from set Π, while labels (corresponding to resources) are from set Λ; a relation $\mathcal{A}$ has the form $\mathcal{S} \times \Lambda \times \mathcal{S}$. The fundamental intuition to obtaining the trace is establishing the relation $\mathcal{A}$ that represents accessibility from a state s to its subsequent state—let s' be this subsequent state. To do this, we exploit the fact that a resource application operates on R_λ in a way that yields P_λ in the next state s'. Let $\mathcal{T}$ be a helper function yielding true if label $\lambda \in \Lambda$ and P_λ can be combined leading to R_λ. We represent as s_Π the propositions describing state $s \in \mathcal{S}$ as a conjunction. E is the formulation of the goal of the edge node where coordination takes place. Establishing $\mathcal{A}$ starting from the initial state $\mathcal{I}$, traversing states of the computation, and eventually reaching the device goal state $\mathcal{G}$ amounts to the following formula encoding the computation:

$$(\mathcal{I}_\Pi \wedge \mathsf{E}) \bigwedge_{0 \leq i < k} \mathcal{T}(\mathsf{s}\Pi_i \wedge \mathsf{E}, \Lambda_{i+1}, \mathsf{s}'\Pi_{i+1} \wedge \mathsf{E}) \bigwedge \mathcal{G}_\Pi \wedge \mathsf{E}. \quad (6)$$

Formula (6) starts with a conjunction of a set of propositions describing the initial state conjuncted with the edge's goals (recall that E itself is a first-order logical formula). Subsequently, it encodes the existence of a computation whose transitions are labeled according to resources Λ. Each state $s \in \mathcal{S}$ of the computation is a conjunction between the edge goal formula E and the propositions describing the state. Finally, a goal state ($\mathcal{G}_\Pi$) is reached while maintaining the satisfaction of the edge goal E. The edge goal acts as a further constraint on every computation state—it must be always fulfilled as resources are invoked. The device's goal is the final state reached, while the edge's goal as well as the resource invocations governs how it is reached.

Note that the index k represents the length of the trace. Formula (6) is true if and only if there exists a computation of length k from state $\mathcal{I}$ to goal state $\mathcal{G}$ of the dLTS $\mathcal{K}$. Notice that the formula is a conjunction of a finite collection of literals, thus, in CNF form. Following the definition of (6), a SAT/SMT solver can be used to check its satisfiability [31] for incremental values of k. The smallest k where the formula is satisfied represents the optimal solution. The respective trace represents the solution sequence of resource invocations.

For our resource coordination purposes, we essentially ask for an assignment that satisfies the constraints of each resource, leading to the fulfillment of the device goal. The values of the transitions make up the coordination plan, consisting of the resource invocations that the requesting device must perform to satisfy its goal. Formally, the plan returned is the concatenation $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_n$ of labels (representing resource invocations) amounting to the trace originating from $\mathcal{I}$ and leading to a goal state $\mathcal{G}$ where the device's goal G is satisfied. Certainly, if there exists no satisfiable solution, a plan cannot be computed. If a plan exists, however, at a minimal length k, there are guarantees about optimality—there is no plan at length less than k that satisfies the goal. Given a coordination problem, computation of a plan in practice can be achieved by employing a SAT/SMT solver, from which a satisfiable assignment of (6) is requested.

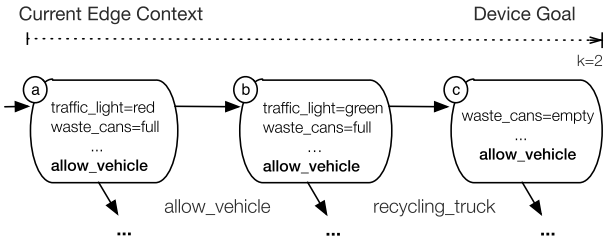

Fig. 5. *dLTS fragment showing an evolution of resources to a device goal. The edge goal is in bold, as a constraint through the states of the computation.*

Fig. 5 shows the dLTS corresponding to our example; state (a) captures the current edge context, where a recycling truck arrives. Recall that the goal of the truck is to empty the waste cans, which it communicates to the local neighborhood edge node. The plan computed at the edge node shows that an invocation of allow_vehicle can enable recycling_truck, which will lead to the satisfaction of the truck's goal. Observe how the presence of allow_vehicle on every computation state acts as a constraint imposed by the edge node for the generated plan. Finally, note that imposing system-wide goals (as illustrated in Section V-B) to edge nodes would imply that each edge is constrained by what occurs within another edge's scope, something that would require a centralized (perhaps priority-based) strategy, which, for the decentralized coordination approach presented, is not desired.

VII. EVALUATION

For evaluating the proposed approach, we developed tool support and a proof-of-concept implementation based on the CVC4 SMT solver [34]. Noting the absence of approaches utilizing SMT solving on the edge, we deployed the prototype on low-powered ARM-based devices representative of edge nodes situated typically close to IoT devices in wide area settings such as smart cities. The technique we advocate for resource matchmaking is based on bounded model checking, a highly computationally expensive operation that is usually performed at design time. However, we bring it to the system runtime. To this end, our evaluation goals target realization and feasibility of our approach for coordinating resources at runtime for the edge-enabled IoT. Concretely, we aim to: 1) investigate feasibility over concrete deployment on low-powered ARM-based edge devices and 2) assess the performance of SMT-based resource matchmaking over hard problem instances.

We present our evaluation setup in Section VII-A, and the experimental results are given in Section VII-B. We conclude with a discussion in Section VII-C.

A. Experiment Setup: Synthesized Resources

Our experiment setup entails: 1) generating a suitable data set and 2) deploying the prototypical framework on low-powered devices that serve as edge nodes. To obtain a suitable matchmaking data set for our experiments, we automatically generate problem instances, each containing: 1) a set of IoT resource specifications; 2) some IoT context assumed to be active when the procedure is invoked; and 3) some resource goal that a device is assumed to have requested. We synthesize matchmaking problem instances, varying the number of resources available, resource preconditions, and the number of operators among them, given a global set of names Π, where $|\Pi| = 100$. Specifically, our experimental data set comprises the specification of a set of 200 problem instances in turn each comprising the following.

1) A set of resources X assumed to be available within an edge scope. Each is modeled per Section V, as $\langle R_\lambda, P_\lambda \rangle$. The cardinality of X ranges from 10 to 50, yielding different problem instances.
2) For each problem instance I, cardinality of R_{λ_i} sets for every resource $i \in I$ ranges from 5 to 15 of parameters, which are randomly combined with a number of operators: $5 \leq |R_\lambda| \leq 15$. Operators are inserted randomly within R_{λ_i}. Resource postconditions are a conjunction of five parameters: $|P_\lambda| = 5$. $R_\lambda, P_\lambda \subseteq \Pi$.
3) A context description, referring to the set of context values when the edge node initiates the coordination process. We assume a conjunction of $|\mathsf{C}| = 20$ such context values, where $\mathsf{C} \subseteq \Pi$.
4) A random device goal G, which is a conjunction of five elements of Π.

From the synthesized problem instances, we select ones that are satisfiable to ensure coverability of the whole process of computing coordination plans presented in Section VI and to reduce noise in the results. Throughout the process, we use Boolean operators only to simplify the automated resource configuration generation, since finding satisfiable instances on random SMT propositions amounts to a random search. Moreover, to ensure uniformity, we consider instances where the optimal plan is found at a bound of 5 (i.e., $k = 5$; see Section VI). Subsequently, we deploy the reasoning machinery on an edge device.

Our prototypical implementation employs the procedure described in Section VI and is deployed on a low-powered ARMv8 R-Pi3 device featuring a 1.2-GHz CPU and 1-GB RAM, serving as the edge node. Given a resource configuration, the edge node's functionality—implemented in Python and C—consists essentially of the following steps: 1) the appropriate bounded model checking formula representation [see (6)] is encoded depending on the problem instance; 2) the CVC4 solver is invoked upon it; and 3) the plan is computed from the satisfiability assignment of the solver. We note that any SMT-LIB compliant SMT solver can be used; the satisfiable assignment from the solver is then used to derive the plan. Functionality is exposed through lightweight REST, with which

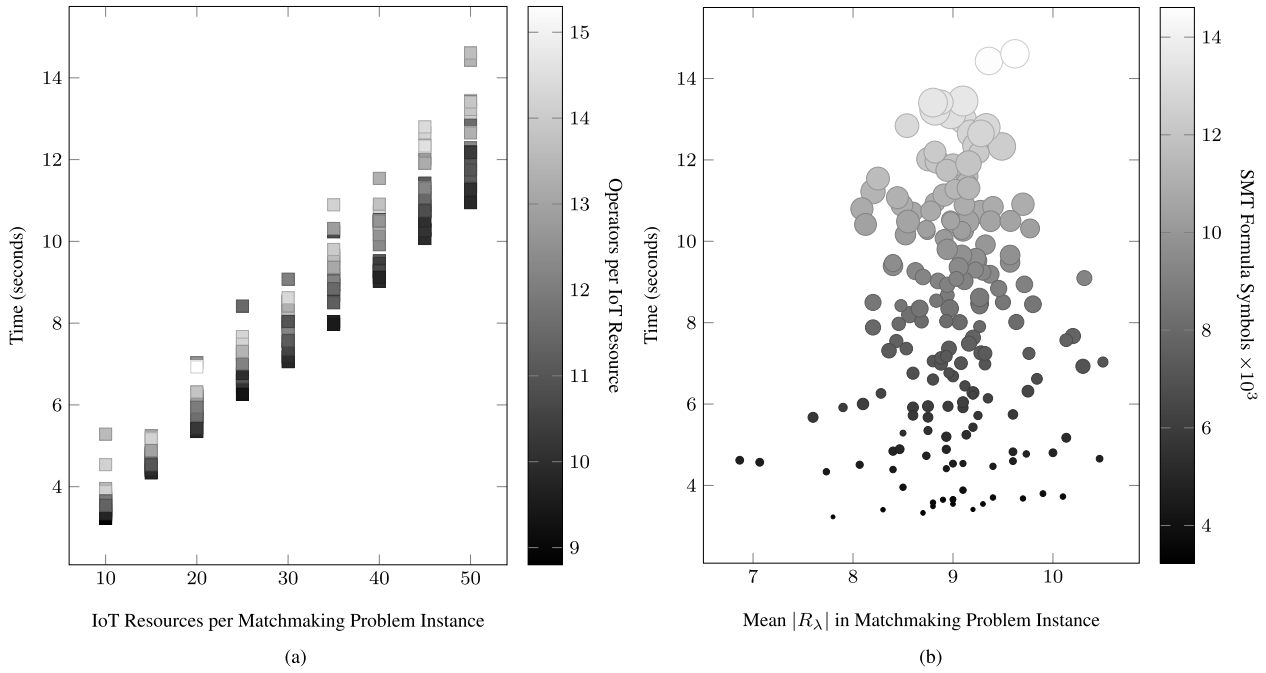

Fig. 6. ***Matchmaking problem instances; coordination time, $|R_\lambda|$, operators, and SMT symbols on an ARMv8 edge device. (a) Resource set cardinality ($|X|$) of matchmaking problem instances over coordination time. Shading in points indicates the average number of operators used in each satisfiable problem instance. (b) Mean resource dependences in matchmaking problem instances over coordination time. Points size indicates the size of the SMT encoding of the respective bounded model checking problem.***

participating devices in the edge scope update context values and request coordination plans. The procedure described is invoked for every requesting device, resulting in the computation of a coordination plan. As we observed in Section VI-C, the coordination plan consists of the resource invocations that the requesting device must perform within the edge scope to satisfy its goal. Subsequently, we evaluate how such instances perform in practice by simulating requests from devices to the edge node. Device requests are drawn from the problem instance data set of the previous step.

B. Experimental Results: Resource Coordination

To obtain the experimental results based on the synthesized data set, we simulate the requests from devices to investigate the performance of the various problem instances and account for the time taken to coordinate the resources in every problem instance. We ignore network overhead, and we report on the total computation performance of the coordination plans, from a request to the plan response by the coordinating edge node.

In Fig. 6(a), the number of IoT resources (X) over time is illustrated—each data point is a single problem instance (i.e., a resources-context-goal configuration). Additionally, one can observe the number of operators used in the required dependences of (every) resource in the configuration (shading in data points). Evidently, the more IoT resources there are in a problem instance, the more time it takes to coordinate. Due to the Boolean satisfiability solving [35] that underlies the matchmaking process, SAT/SMT problem instances with a different number of operators perform differently, although the number of resources is kept constant [i.e., within a vertical line in Fig. 6(a)]. For example, one can observe that in the vertical line denoting 25 IoT resources, some problem instance utilizing 14 operators per (every) resource performs better than some particular problem instance with ten operators. Hardness of SAT/SMT satisfiability [36] is beyond the scope of this paper. Due to the synthesized nature of our evaluation data set, we did not consider edge goals as those would be defined per application. However, those would not affect the results significantly due to the small expected size that their encoding would add as an overhead.

Fig. 6(b) captures the mean $|R_\lambda|$ size per matchmaking problem instance across time. Each data point is a single problem instance—the same resources-context-goal configurations of Fig. 6(a). The size of the resulting SAT/SMT formulas in symbols [as per (6)] corresponding to the coordination problem encoding is represented by the point size. A number of symbols within formulas range from 4k to 14k. Naturally, as formula size increases, so does the coordination time. We can observe that again, certain problem instances with small average cardinality $|R_\lambda|$ lead to hard instances, and vice versa. However, the formula size is a strong indicator of coordination time.

Based on the above-mentioned results, a system designer can obtain insights depending on her particular problem setting. Within a typical design process, a designer

requires knowledge of the performance of the system due to the definition of some service-level agreement (SLA). Our evaluation results show that by selecting a number of IoT resources [e.g., 20 resources in Fig. 6(a)] and an average number of operators per resource dependences (e.g., 10), a matchmaking performance at least 5 s is to be expected. Then, formula size and average $|R_\lambda|$ can also give an indication of the expected time.

C. Discussion

We have demonstrated that by using our coordination framework, coordination of IoT resources in an edge setting in a dependable manner can be performed. Furthermore, we showed that our technique based on SMT-based bounded model checking at the edge is performant and feasible for realistic problem sizes even on low-powered ARM-based devices. We especially note that our resource coordination technique guarantees correct and optimal results due to its satisfiability foundations.

We showed the performance based on synthesized IoT resource problem instances. Our results are actionable since a system designer can estimate the performance of coordination based on the problem space induced in her particular setting. This, combined with testing resource configurations prior to deployment, can drive design decisions during system development. Essentially, given a number of IoT resources, propositions, and operators per resource [see Fig. 6(a) and (b)], one can estimate a time that coordination computation can be achieved. The type of operators used within specification as well as the formula structure obviously affects satisfiability. We plan to investigate what type and mix of operators occur in practice in IoT resources specification and construct the guidelines and metrics relevant to the resource encoding arising from our coordination technique.

Several assumptions inherent in our approach must be further investigated. For evaluation purposes, we considered a coordination plan of length 5; however, there may exist settings where a higher or lower plan length is desired. Moreover, implementing the approach with optimality in mind entails finding plans first on plan length 1, then if none is found on plan length 2, etc. To optimize this stepwise search, the SMT problem encoded can be incrementally introduced to a solver that makes use of past unfoldings to possibly find satisfiable solutions faster. We identify this as future work.

Moreover, the temporal aspects of both the specification and the overall process must be investigated. First, the planning time plus its execution (i.e., the device invoking the resources described in the plan) must be faster than the rate of change of the environment—as such, longer plans may not be advisable, and k should be largely kept small [37]. Second, temporal aspects regarding resource invocations are not captured in the model. Algebraic operations upon countable parameters would also be useful, as invocation of a resource microservice of a battery-powered actuator might consume energy. In our example for instance, irrigation in the park may take time. We identify integrating temporal aspects both regarding the model as well as planning and execution as a significant avenue of future work.

Operationalization of our framework could also benefit from domain-specific adjustments and heuristics. For example, previous plans may be stored (e.g., memoized) to avoid computing them again. The depth of the solution search may be adjusted depending on the current edge computational load or other factors. On the problem level, grouping IoT resources in an ontology can allow the underlying solver to disregard irrelevant or unsatisfiable solutions faster, thus rendering our approach capable to consider a higher number and more complex IoT resources. Finally, we note the absence of approaches utilizing SAT/SMT solving at the edge and underline the opportunities that this brings for dependable edge-enabled IoT settings.

VIII. RELATED WORK

We presented a methodology and technical framework to engineering resource coordination for the edge-enabled IoT, thus touching upon several research areas. Consequently, we classify the related work into three categories. First, we discuss key approaches in the conception of resources as services within IoT. Then, we review related techniques on service composition, as they apply to IoT. Finally, we discuss the related engineering approaches from the domain of self-adaptive systems, framing our approach within the overall software engineering domain.

A. Resources as Services Within IoT

The prevalence of Internet-connected devices featuring various actuation and sensing capabilities provides new means for the development of composite software systems. So far, cloud computing has been seen as a key component for the development, deployment, and coordination of IoT collectives.

In recent years, the paradigm of service-oriented architecture (SOA) [9], [38] has received considerable attention in the field of IoT. Spiess *et al.* [39] proposed an architecture for effective integration of IoT in enterprise services, where they are used to implement business processes. Since the services are offered in a device level with frequent changes, a traditional business process language, such as BPEL, is not built to support such dynamics. As a result, an extended version of BPEL is provided to model business processes at design time, which supports the dynamic changes of services during process execution. Furthermore, to satisfy the application needs or in response to unforeseen context changes, it is possible to remotely deploy new services during runtime. Meyer *et al.* [40] investigated how an "IoT device" component and its native services can be expressed as a resource in an IoT-aware process model. Cheng *et al.* [41] proposed a situation-aware IoT coordination platform based

on the event-driven SOA paradigm. The proposed system architecture effectively utilizes SOA and EDA paradigms—SOA is used to resolve interoperability issues among heterogeneous services and physical entities, while EDA is used to address the problem of the cross-business domain. Furthermore, Zhang *et al.* [42] presented an event-driven SOA for IoT services. Sarkar *et al.* [43] proposed a layered and distributed architecture for IoT, which overcomes most of the obstacles in the process of large-scale expansion of IoT.

In pervasive environments, selecting appropriate resources and services that satisfy user's requirements is a challenge. Due to their dynamic nature, efficient resource discovery is essential in order to achieve wide user acceptance. A significant number of works within service discovery have been focused on context-based approaches. Butt *et al.* [44] provided a service selection technique to offer the appropriate service to a user application depending on the available context information. Rasch *et al.* [45] proposed a proactive service discovery approach for pervasive environments, described by a formal context model that effectively captures the dynamics of context and the relationship between services and context. Wang and Chow [46] proposed an architecture for service discovery in smart cities, focusing on taking a set of devices around a citizen and proactively producing a list of services that surround the user using his preferences. Jin *et al.* [10] described device, resource, and service as the three core concepts in a model that specifies relationships among them. Moreover, the quality of service attributes is defined, which reflects the features of physical services. Yang and Li [47] proposed an efficient strategy from the perspective of sensory and data selections and aggregation, with genetic algorithms as the global optimization method. Since we adopt the concept of XaaS abstraction to uniformly represent physical things, hardware and software resources as microservices, such discussed works are relevant to our proposed approach.

Well-known approaches adopt semantic web technologies and matching techniques for effective service discovery. Zhu and Meng [48] designed service discovery in pervasive computing using the description language OWL-S, matching services according to their category, Input/output parameters, and QoS. Mokhtar *et al.* [49] supported efficient, semantic, context- and QoS-aware service discovery [50] on top of the existing service discovery protocols (SDPs) [51]—this operates at a higher, semantic abstraction level, and is thus independent of the specific underlying SOA technology employed. In addition, a language for semantic service description covers both functional and non-functional service characteristics as well as a set of conformance relations and prescribes the way for applying them in order to perform service matching. Approaches related to the semantic web technologies used for service discovery are also relevant to our proposed approach. Since semantic service description covers both functional and non-functional, they can be included also to the methodology that we propose for specifying resources.

B. Service Composition and IoT

As SOA becomes widely used, providing the right services that satisfy a user's goal is becoming a big challenge in IoT environments. Due to dynamicity, heterogeneity, and function constraints, IoT services differ from traditional services. In addition, IoT services are related more to the physical world by sensing state and inducing operations that will cause a state change. A great number of approaches have been proposed to deal with such service composition; we employ a SAT-based technique similar to [52], where the semantic aspect is considered, enabling the composition engine to identify correct, complete and optimal candidates as a solution. However, we extend it to SMT, we use linear integer arithmetic and first-order formulas and deploy on resource-constrained edge devices instrumented at runtime. Mayer *et al.* [53] presented a consistency-based service composition approach that serves as a unified platform. The proposed framework is based on a declarative constraint language to express user requirements, process constraints, and service profiles on a conceptual level and also on the instance level. Pistore *et al.* [54] proposed a solution for automated composition at the process level using OWL-S. A composition considers that executing a Web service requires interactions that may involve different sequential, conditional, and iterative steps. A process-level description of the composite service is generated by using each individual description of services. However, the proposed solution does not consider selecting the services that take part in the composition.

An architectural approach to enable the automated formation and adaptation of emergent configurations (ECs) in the IoT have been proposed in [55]. An EC is formed by a set of things, with their services, functionalities, and applications, to realize a user goal. ECs are adapted in response to (un)foreseen context changes, e.g., changes in available things or due to changing or evolving user goals. Hussein *et al.* [56] proposed a model-driven approach to ease the development of adaptive IoT systems. A design model is specified based on the system requirements as well as the system functionality and adaptations. Furthermore, it is used to generate the system implementation, transformed into an IoT platform-specific model. This model is used for generating code and a deployment to a hardware platform. After adaption is triggered, the system changes its state based on the designed model. Urbieta *et al.* [57] proposed an adaptive service composition framework. The framework is based on an abstract service model representing services and user tasks in terms of their signature, specification, and conversation. Xinming and Yan [58] proposed a service mining scheme based on semantic for IoT to provide users with interesting composite services. Service composition is achieved by combining and recommending to users according to the calculation of service similarity—however, not including service composition QoS.

Ciortea *et al.* [59] proposed a decentralized approach to IoT mash-up composition that considers flexibility and responsiveness of resulting applications. Goal-driven software agents are equipped with precompiled plans, which cooperate with one another through sociotechnical networks (STNs) to compose IoT mash-ups at runtime in pursuit of their goals. Various IoT devices are modeled as agents based on their capabilities. Agents are goal-driven and rely on precompiled plans that specify how to achieve their goals. Whenever the goal cannot be fulfilled by a single agent, agents cooperate with one another through STNs to compose mash-ups that achieve the goals. In contrast, we synthesize plans at runtime and utilizing available resources opportunistically from the runtime edge context. Mayer *et al.* [60] proposed a service composition system that enables the goal-driven configuration of smart environments for end users by combining semantic metadata and reasoning with a visual modeling tool—instead of using predefined service mash-ups, creation of them in a dynamic manner fulfills the desired user goal. This dynamicity is similar to our coordination approach. Such flexibility is achieved by using embedded semantic API descriptions. Hence, service mash-ups can adapt to dynamic environments and are fault-tolerant.

C. Self-Adaptive Systems and IoT

Self-adaptive software becomes an inseparable part of systems characterized by uncertain environments, evolving requirements, and unexpected failures. In order to meet strict functional and non-functional requirements in applications within diverse areas, Calinescu *et al.* [61] proposed a methodology and instantiation of dynamic safety cases which allows adjusting the system during execution while providing the intended functionality and its requirements. Marrella *et al.* [62] proposed a model and prototype process management system featuring a set of techniques providing support for automated adaptation of knowledge-intensive processes at runtime. Such techniques are able to automatically adapt and recover process instances when an exception occurs and without the intervention of domain experts at runtime. Chen *et al.* [63] proposed a runtime model-based approach to IoT application development. The initial step toward the proposed approach is considering that sensor devices are abstracted as runtime models that are automatically connected with the corresponding systems. Based on the application scenario, a customized model is constructed and the synchronization between the model and the sensor device is achieved through model transformation. As a result, the application logic is mapped and executed on the customized model after some definitions are given, such as group of metamodels, mapping rules, and model-level programs.

In the context of self-management architectures, a significant number of generic approaches have been proposed. Kramer and Magee [64] proposed a three-layer reference model to support automatic (re)configuration of self-managed systems, consisting of a component control layer, a change management layer, and a goal management layer. The component layer is responsible to provide change management that reconfigures the software components, while the change management generates the plans to achieve system goals. An overall model relies on a set of plans which aims to achieve the desired system goals. Whenever new goals are introduced to the system, the change management layer is responsible to generate new plans for achieving desired goals. Thus, we follow this methodology essentially targeting low-powered ARM-based edge computers for deployment. In addition, the resource coordination facilities we provide are dependable. Weyns *et al.* [65] proposed an architecture-based adaptation approach to solve the concrete problem of automating the management of IoT. The software system utilizes a feedback loop that employs models@runtime and statistical techniques to reason about the system and induce adaptation to ensure the required goals.

Software systems are deployed in dynamic environments that change over time and often have to adapt to the changing conditions in order to meet system goals. The well-known approaches have been developed for runtime monitoring for different kinds of systems. Seiger *et al.* [66] presented an approach for enabling self-adaptive workflows based on the Monitor, Analyze, Plan and Execute on a Knowledge Base (MAPE-K) control loop for self-adaptive workflows in cyber-physical systems. The proposed approach within MAPE-K loop monitors and analyzes the real-world effects through sensor and context data, which is used to check for faulty errors in the physical world. If an inconsistency between the sensed physical world and the assumed cyber world can be detected, a compensation strategy is chosen and the adapted process is executed. Cailliau and van Lamsweerde [67] proposed obstacle-driven runtime adaptation techniques for an increased satisfaction of probabilistic system goals. The approach is based on the MAPE cycle and relies on a model where goals and obstacles are refined and specified in a probabilistic manner. However, in contrast to the proposed approach, we guarantee the optimality and correctness of generated coordination plans—we identify quantitative extensions to our goal modeling as future work. The resource coordination facilities we provide are dependable because if there is a solution to a resource coordination problem for a device, the technique we utilize will provide a plan for it, and the plan will be optimal. This is in contrast to other approaches utilizing other coordination techniques such as based on AI [20]–[22].

IX. CONCLUSION

Software components within pervasive IoT systems make use of resources which can be various computational capabilities, including sensing or actuation end points. Components do not live in isolation and must be able to

coordinate with others to fulfill their goals. Edge computers placed near end devices can be leveraged for control—providing resource management for devices within their active context. To this end, we proposed a methodology and technical framework for engineering resource coordination at runtime, tailored for the decentralized, pervasive systems of today. We adopted goal modeling to capture objectives within the IoT and used bounded model checking as the foundational technique to compute coordination plans that satisfy device goals. This occurs opportunistically at runtime, without any knowledge about the operational status or presence of resources in the system at the system's design time, but always in accordance with the edge's own goals. Our technical framework exhibits dependability guarantees regarding optimality and correctness of generated coordination plans and is realizable on edge nodes deployed on low-powered ARM-based edge devices.

We believe coordination at the edge as presented paves the way for situating control logic close to end devices, leading to increased decentralization in edge-based systems. We plan to investigate dealing with conflicting goals, where edge nodes or devices have requirements that overlap but do not agree [68] and may require negotiation to resolve. Although goals in our approach were assumed to be functional, if nonfunctional requirements are considered, tradeoffs may need to be made with respect to, e.g., cost, performance, energy, and so on. We identify integrating temporal aspects both regarding the model as well as planning and execution performance as a significant avenue of future work. Resources may be countable, reflecting some limited availability such as a cost—this requires extending the encoding. Moreover, resource invocation timings can be captured in the model, as countable parameters would be useful for quantitative requirements specification, related to, e.g., device SLAs. Operational aspects that need to be investigated regard that the coordination process must be faster than the rate of change of the environment. Finally, communication and operational aspects were not treated within the dependable runtime coordination approach presented. Considering the perspective of a complete realization, such distributed systems aspects need to be treated. ■

REFERENCES

[1] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," in *Internet Things*. Amsterdam, The Netherlands: Elsevier, 2016, pp. 61–75.

[2] C. Tsigkanos, S. Nastic, and S. Dustdar, "Towards resilient Internet of Things: Vision, challenges, and research roadmap," in *Proc. 39th IEEE Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Dallas, TX, USA, Jul. 2019, pp. 1–11.

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generat. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[4] V. R. Lesser and D. D. Corkill, "Functionally accurate, cooperative distributed systems," in *Readings in Distributed Artificial Intelligence*. Amsterdam, The Netherlands: Elsevier, 1988, pp. 295–310.

[5] C. Tsigkanos, T. Kehrer, and C. Ghezzi, "Modeling and verification of evolving cyber-physical spaces," in *Proc. ACM 11th Joint Meeting Found. Softw. Eng.*, 2017, pp. 38–48.

[6] C. Tsigkanos, L. Nenzi, M. Loreti, M. Garriga, S. Dustdar, and C. Ghezzi, "Inferring analyzable models from trajectories of spatially-distributed Internet-of-Things," in *Proc. 1st IEEE/ACM Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst., (SEAMS@ICSE)*, Montreal, QC, Canada, May 2019.

[7] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton, "Reference model for service oriented architecture 1.0," *OASIS Standard*, vol. 12, p. 18, Feb. 2006.

[8] A. Bouguettaya *et al.*, "A service computing manifesto: The next 10 years," *Commun. ACM*, vol. 60, no. 4, pp. 64–72, 2017.

[9] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of Things: Discovery, query, selection, and on-demand provisioning of Web services," *IEEE Trans. Services Comput.*, vol. 3, no. 3, pp. 223–235, Jul./Sep. 2010.

[10] X. Jin, S. Chun, J. Jung, and K.-H. Lee, "IoT service selection based on physical service model and absolute dominance relationship," in *Proc. IEEE 7th Int. Conf. Service-Oriented Comput. Appl. (SOCA)*, Nov. 2014, pp. 65–72.

[11] N. Li, C. Tsigkanos, Z. Jin, S. Dustdar, Z. Hu, and C. Ghezzi, "POET: Privacy on the edge with bidirectional data transformations," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, Kyoto, Japan, Mar. 2019, pp. 1–10.

[12] J. Ren, H. Guo, C. Xu, and Y. Zhang, "Serving at the edge: A scalable IoT architecture based on transparent computing," *IEEE Netw.*, vol. 31, no. 5, pp. 96–105, Aug. 2017.

[13] C. Barrett and C. Tinelli, *Satisfiability Modulo Theories*. Springer, 2018.

[14] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded model checking," *Adv. Comput.*, vol. 58, pp. 117–148, 2003.

[15] J. Rao and X. Su, "A survey of automated Web service composition methods," in *Proc. Int. Workshop Semantic Web Services Web Process Composition*. Berlin, Germany: Springer, 2004, pp. 43–54.

[16] A. Ankolekar *et al.*, "DAML-S: Web service description for the semantic Web," in *Proc. Int. Semantic Web Conf.* Berlin, Germany: Springer, 2002, pp. 348–363.

[17] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web services Web: An introduction to SOAP, WSDL, and UDDI," *IEEE Internet Comput.*, vol. 6, no. 2, pp. 86–93, Mar. 2002.

[18] D. L. McGuinness and F. Van Harmelen, "Owl Web ontology language overview," *W3C Recommendation*, vol. 10, no. 10, p. 2004, 2004.

[19] E. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods Syst. Des.*, vol. 19, no. 1, pp. 7–34, 2001.

[20] F. Alkhabbas, R. Spalazzese, and P. Davidsson, "ECo-IoT: An architectural approach for realizing emergent configurations in the Internet of Things," in *Proc. Eur. Conf. Softw. Archit.* Cham, Switzerland: Springer, 2018, pp. 86–102.

[21] J. Hoffmann and B. Nebel, "The FF planning system: Fast plan generation through heuristic search," *J. Artif. Intell. Res.*, vol. 14, pp. 253–302, May 2001.

[22] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory & Practice*. Amsterdam, The Netherlands: Elsevier, 2004.

[23] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design*. Upper Saddle River, NJ, USA: Prentice-Hall, 2005.

[24] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decade's overview," *Inf. Sci.*, vol. 280, pp. 218–238, Oct. 2014.

[25] S. De, P. Barnaghi, M. Bauer, and S. Meissner, "Service modelling for the Internet of Things," in *Proc. IEEE Federated Conf. Comput. Sci. Inf. Syst. (FedCSIS)*, Sep. 2011, pp. 949–955.

[26] M. S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard, "Reconciling system requirements and runtime behavior," in *Proc. 9th Int. Workshop Softw. Specification Design*, 1998, p. 50.

[27] A. Van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Hoboken, NJ, USA: Wiley, 2009.

[28] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *Proc. 12th Int. Conf. Model Driven Eng. Lang. Syst. (MoDELS)*, 2009, pp. 468–483.

[29] S. Liaskos, S. A. McIlraith, S. Sohrabi, and J. Mylopoulos, "Integrating preferences into goal models for requirements engineering," in *Proc. 18th IEEE Int. Requirements Eng. Conf.*, Sep./Oct. 2010, pp. 135–144.

[30] A. V. Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*, vol. 10, Chichester, U.K.: Wiley, 2009.

[31] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proc. ACM 36th Annu. ACM/IEEE Design Autom. Conf.*, Jun. 1999, pp. 317–320.

[32] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[33] N. Een, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *Proc. Int. Conf. Formal Methods Comput.-Aided Design*, Austin, TX, USA: FMCAD Inc., 2011, pp. 125–134.

[34] C. Barrett *et al.*, "CVC4," in *Proc. 23rd Int. Conf. Comput. Aided Verification (CAV)*, in Lecture Notes in Computer Science, vol. 6806, G. Gopalakrishnan and S. Qadeer, Eds. Snowbird, Utah: Springer, Jul. 2011, pp. 171–177.

[35] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham, "Understanding random SAT: Beyond the clauses-to-variables ratio," in *Proc.*

Int. Conf. Principles Pract. Constraint Program. Berlin, Germany: Springer, 2004, pp. 438–452.
[36] C. Ansótegui, M. L. Bonet, J. Levy, and F. Manyà, "Measuring the hardness of SAT instances," in *Proc. AAAI*, vol. 8, 2008, pp. 222–228.
[37] C. Tsigkanos, L. Pasquale, C. Ghezzi, and B. Nuseibeh, "On the interplay between cyber and physical spaces for adaptive security," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 3, pp. 466–480, May/Jun. 2018.
[38] J.-S. Leu, C.-F. Chen, and K.-C. Hsu, "Improving heterogeneous SOA-based IoT message stability by shortest processing time scheduling," *IEEE Trans. Services Comput.*, vol. 7, no. 4, pp. 575–585, Oct. 2014.
[39] P. Spiess *et al.*, "SOA-based integration of the Internet of Things in enterprise services," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2009, pp. 968–975.
[40] S. Meyer, A. Ruppen, and C. Magerkurth, "Internet of Things-aware process modeling: Integrating IoT devices as business process resources," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.* Berlin, Germany: Springer, 2013, pp. 84–98.
[41] B. Cheng, D. Zhu, S. Zhao, and J. Chen, "Situation-aware IoT service coordination using the event-driven SOA paradigm," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 2, pp. 349–361, Jun. 2016.
[42] Y. Zhang, L. Duan, and J. L. Chen, "Event-driven SOA for IoT services," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jun./Jul. 2014, pp. 629–636.
[43] C. Sarkar, A. U. N. S. N., R. V. Prasad, A. Rahim, R. Neisse, and G. Baldini, "DIAT: A scalable distributed architecture for IoT," *IEEE Internet Things J.*, vol. 2, no. 3, pp. 230–239, Jun. 2015.
[44] T. A. Butt, I. Phillips, L. Guan, and G. Oikonomou, "Adaptive and context-aware service discovery for the Internet of Things," in *Internet Things, Smart Spaces, Next Gener. Netw.* Berlin, Germany: Springer, 2013, pp. 36–47.
[45] K. Rasch, F. Li, S. Sehic, R. Ayani, and S. Dustdar, "Context-driven personalized service discovery in pervasive environments," *World Wide Web*, vol. 14, no. 4, pp. 295–319, 2011.
[46] E. Wang and R. Chow, "What can i do here? IoT service discovery in smart cities," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2016, pp. 1–6.
[47] Z. Yang and D. Li, "IoT information service composition driven by user requirement," in *Proc. IEEE 17th Int. Conf. Comput. Sci. Eng. (CSE)*, Dec. 2014, pp. 1509–1513.
[48] Y.-A. Zhu and X.-H. Meng, "A framework for service discovery in pervasive computing," in *Proc. IEEE 2nd Int. Conf. Inf. Eng. Comput. Sci. (ICIECS)*, Dec. 2010, pp. 1–4.
[49] S. Ben Mokhtar, D. Preuveneers, N. Georgantas, V. Issarny, and Y. Berbers, "EASY: Efficient semantic service discovery in pervasive computing environments with QoS and context support," *J. Syst. Softw.*, vol. 81, no. 5, pp. 785–808, 2008.
[50] S. Ben Mokhtar, A. Kaul, N. Georgantas, and V. Issarny, "Efficient semantic service discovery in pervasive computing environments," in *Proc. ACM/IFIP/USENIX Int. Conf. Middleware*. New York, NY, USA: Springer-Verlag, 2006, pp. 240–259.
[51] F. Zhu, M. W. Mutka, and L. M. Ni, "Service discovery in pervasive computing environments," *IEEE Pervasive Comput.*, vol. 4, no. 4, pp. 81–90, Oct./Dec. 2005.
[52] H. Kil and W. Nam, "Semantic Web service composition via model checking techniques," *Int. J. Web Grid Services*, vol. 9, no. 4, pp. 339–350, 2013.
[53] W. Mayer, R. Thiagarajan, and M. Stumptner, "Service composition as generative constraint satisfaction," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2009, pp. 888–895.
[54] M. Pistore, P. Roberti, and P. Traverso, "Process-level composition of executable Web services: 'On-the-fly' versus 'once-for-all' composition," in *Proc. Eur. Semantic Web Conf.* Berlin, Germany: Springer, 2005, pp. 62–77.
[55] F. Alkhabbas, R. Spalazzese, and P. Davidsson, "ECo-IoT: An architectural approach for realizing emergent configurations in the Internet of Things," in *Proc. Eur. Conf. Softw. Archit.* Cham, Switzerland: Springer, 2018, pp. 86–102.
[56] M. Hussein, S. Li, and A. Radermacher, "Model-driven development of adaptive IoT systems," in *Proc. 4th Int. Workshop Interplay Model-Driven Compon.-Based Softw. Eng. (ModComp) Workshop*, 2017, p. 20.
[57] A. Urbieta, A. González-Beltrán, S Ben Mokhtar, M A. Hossain, and L. Capra, "Adaptive and context-aware service composition for IoT-based smart cities," *Future Gener. Comput. Syst.*, vol. 76, pp. 262–274, Nov. 2017.
[58] X. Li and Y. Sun, "A service mining scheme based on semantic for Internet of Things," *Chin. J. Electron.*, vol. 23, no. 2, pp. 236–242, 2014.
[59] A. Ciortea, O. Boissier, A. Zimmermann, and A. M. Florea, "Responsive decentralized composition of service mashups for the Internet of Things," in *Proc. ACM 6th Int. Conf. Internet Things*, 2016, pp. 53–61.
[60] S. Mayer, R. Verborgh, M. Kovatsch, and F. Mattern, "Smart configuration of smart environments," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 3, pp. 1247–1255, Mar. 2016.
[61] R. Calinescu, D. Weyns, S. Gerasimou, M. U. Iftikhar, I. Habli, and T. Kelly, "Engineering trustworthy self-adaptive software with dynamic assurance cases," *IEEE Trans. Softw. Eng.*, vol. 44, no. 11, pp. 1039–1069, Nov. 2018.
[62] A. Marrella, M. Mecella, and S. Sardina, "Intelligent process adaptation in the SmartPM system," *ACM Trans. Intell. Syst. Technol.*, vol. 8, no. 2, p. 25, 2016.
[63] X. Chen, A. Li, X. Zeng, W. Guo, and G. Huang, "Runtime model based approach to IoT application development," *Frontiers Comput. Sci.*, vol. 9, no. 4, pp. 540–553, 2015.
[64] J. Kramer and J. Magee, "Self-managed systems: An architectural challenge," in *Proc. Future Softw. Eng.*, 2007, pp. 259–268.
[65] D. Weyns, M. U. Iftikhar, D. Hughes, and N. Matthys, "Applying architecture-based adaptation to automate the management of Internet-of-Things," in *Proc. Eur. Conf. Softw. Archit.* Cham, Switzerland: Springer, 2018, pp. 49–67.
[66] R. Seiger, S. Huber, P. Heisig, and U. Aßmann, "Toward a framework for self-adaptive workflows in cyber-physical systems," *Softw. Syst. Model.*, vol. 18, no. 2, pp. 1117–1134, 2017.
[67] A. Cailliau and A. van Lamsweerde, "Runtime monitoring and resolution of probabilistic obstacles to system goals," in *Proc. IEEE/ACM 12th Int. Symp. Softw. Eng. Adapt. Self-Manag. Syst. (SEAMS)*, May 2017, pp. 1–11.
[68] B. Nuseibeh, J. Kramer, and A. Finkelstein, "A framework for expressing the relationships between multiple views in requirements specification," *IEEE Trans. Softw. Eng.*, vol. 20, no. 10, pp. 760–773, Oct. 1994.

ABOUT THE AUTHORS

Christos Tsigkanos received the B.Sc. degree in computer science from the University of Athens, Athens, Greece, the M.Sc. degree in software engineering from the University of Amsterdam, Amsterdam, The Netherlands, and the Ph.D. degree from the Politecnico di Milano, Milan, Italy, in 2017, under the supervision of Prof. C. Ghezzi. His Ph.D. dissertation was titled "Modelling and Verification of Evolving Cyber-Physical Spaces."

He was a Postdoctoral Researcher with the Politecnico di Milano. He is currently a Researcher with the Distributed Systems Group, TU Wien, Vienna, Austria. His current research interests include the intersection of dependable systems, formal aspects of software engineering, security and privacy in distributed, self-adaptive and cyber–physical systems, requirements engineering, and formal verification.

Ilir Murturi received the M.Sc. degree in computer engineering from the Faculty of Electrical and Computer Engineering, University of Prishtina, Pristina, Kosovo. He is currently working toward the Ph.D. degree in edge computing under the supervision of Prof. S. Dustdar at the Distributed Systems Group, TU Wien, Vienna, Austria.

His current research interests include the Internet of Things, edge computing, crowdsourcing, privacy, and smart cities.

Schahram Dustdar (Fellow, IEEE) was an Honorary Professor of information systems with the University of Groningen, Groningen, The Netherlands, from 2004 to 2010, a Visiting Professor with the University of Seville, Seville, Spain, from 2016 to 2017, and a Visiting Professor with the University of California at Berkeley, Berkeley, CA, USA, in 2017. He is currently a Professor of computer science with the Distributed Systems Group, TU Wien, Vienna, Austria.

Dr. Dustdar is an elected member of the Academia Europaea, where he is also the Chairman of the Informatics Section. He serves on the Editorial Board of IEEE INTERNET COMPUTING and the *IEEE Computer Magazine*. He was a recipient of the ACM Distinguished Scientist Award in 2009, the IBM Faculty Award in 2012, and the IEEE TCSVC Outstanding Leadership Award for outstanding leadership in services computing in 2018. He is also the Co-Editor-in-Chief of the *ACM Transactions on Internet of Things* and the Editor-in-Chief of *Computing* (Springer). He is also an Associate Editor of the IEEE TRANSACTIONS ON SERVICES COMPUTING, the IEEE TRANSACTIONS ON CLOUD COMPUTING, the *ACM Transactions on the Web*, and the *ACM Transactions on Internet Technology*.