IoT-CANE: A Unified Knowledge Management System for Data-Centric Internet of Things Application Systems

Yinhao Li[a,*], Awatif Alqahtani[a], Ellis Solaiman[a], Charith Perera[a], Prem Prakash Jayaraman[b], Rajkumar Buyya[c], Graham Morgan[a], Rajiv Ranjan[d,a]

[a]*Newcastle University, Newcastle upon Tyne, UK*
[b]*Swinburne University of Technology, Victoria, Australia*
[c]*The University of Melbourne, Victoria, Australia*
[d]*China University of Geosciences (Wuhan), Hubei, China*

Abstract

Identifying a suitable configuration of devices, software and infrastructures in the context of user requirements is fundamental to the success of delivering IoT applications. As possible configurations could be large in number and not all configurations are valid, a configuration knowledge representation model can provide ready-made configurations based on IoT requirements. Combining such a model within the context of a given user-oriented scenario, it is possible to automate the recommendation of solutions for deployment and long-time evolution of IoT applications. However, in the context of Cloud/Edge technologies, that may themselves exhibit significant configuration possibilities that are also dynamic in nature, a more unified approach is required. We present IoT-CANE (Context Aware recommendatioN systEm) as such a unified approach. IoT-CANE embodies a unified conceptual model capturing configuration, constraint and infrastructure features of Cloud/Edge together with IoT devices. The success of IoT-CANE is evaluated through an end-user case study.

Keywords: Internet of Things, knowledge representation, context-aware, recommender system, Ripple Down Rules, resource configuration, configuration management

*Corresponding author

1. Introduction

The Internet of things (IoT) is the idea of things (devices) that are locatable, readable and recognizable and controllable through using the Internet [1, 2, 3]. In IoT a number of things and resources (Edge and/or Cloud) combine to provide services that form the basis for many different types of applications that are commonly referred to as smart (e.g., smart healthcare, smart homes, smart buildings, smart manufacturing, smart agriculture, smart traffic). "Smart" is a way of categorizing those applications where users can discover, query and employ different IoT entities on demand in real-time without such entities requiring further human intervention in their development to achieve the desired outcomes. Numerous entities are developed by manufacturers for use within IoT infrastructures. These entities are not only physical, such as sensors and actuators, but also virtual, like social media (e.g., Facebook, Twitter, MySpace). The heterogeneous large-scale data from such physical and virtual entities (e.g., [4], [5], [6], [7],) raises a challenge of unified resource configuration knowledge representation and high performance data processing [8]. Therefore, we need to incorporate the digital world and physical worlds in IoT ecosystems. In order to allow this level of interoperability, it is important to define the services supplied by these physical and virtual entities in a homogeneous way [9]. More specifically, the requirement of developing a novel and unified conceptual model to represent the knowledge and configuration information of each entity in the IoT field is necessary. However, given the propriety nature of manufacturer platforms coupled with the variety of propriety Cloud/Edge standards this is a challenging problem.

In order to sketch the problem domain from a user's viewpoint, consider a scenario of a manager creating a smart building application at minimal financial cost. The manager purchases low priced IoT devices from different manufacturers (e.g., temperature sensors, motion sensors, humidity sensors, Raspberry Pi, enabling gateways). We may assume the manager does not have knowledge of Cloud and Edge resource configuration management and deployment. Therefore, the manager needs to find some IoT application solution provider to offer IoT resource configuration management and deployment solutions given the IoT purchases and existing IT infrastructures. However, the majority of IoT application providers offer expensive solutions that predominantly favour existing services within their own service lists which include only IoT devices supported by their software. The ability to

combine any IoT device produced by different manufacturers would be difficult as there may well exist propriety Application Programming Interfaces (APIs). This increases the difficulty of service deployment from a single IoT application solution provider as additional development may be required to provide API wrappers for devices outside a provider's standard IoT solution list. This is a significant problem as there are more than 325 available APIs for IoT programming listed in the ProgrammableWeb website [10]. Programming requirements from such a diverse range of APIs and associated IoT devices brings new challenges in configuration management and deployment of IoT applications resulting in increasing financial cost to the IoT user.

In another scenario, a householder purchases a new smart camera to enhance their smart home application with an intrusion detection service. They would like to connect a camera to their smart home environment by establishing a connection between the camera and a gateway to collect graphical data. Another IoT entity in the smart home application can capture the graphic data and may detect any intrusion via real-time image analysis. The householder may have already implemented the smart home with the help of a professional IoT solution provider firm. Unfortunately, it may not be affordable or practical to seek such help when enhancing the current smart home system with additional new IoT devices to further enhance functionality. Discovering and adding a new device or a new service into an existing IoT application becomes a challenge for the end user, who would probably have little or no knowledge of IoT infrastructures beyond the commercial instructions on the devices themselves. For example, the new data format for the images originating from the newly purchased smart camera may be new for this smart home application and require additional integration technologies and updates to the existing IoT infrastructure. As Smart technology expands we expect many IoT application users will meet diverse problems in IoT resource configuration management and deployment. Furthermore, due to evolving IoT development diversity, the challenge of meeting this problem needs to be managed in a structured way.

To address the challenges of increased diversity and heterogeneity within IoT solution provision we present a unified configuration management and recommendation system that spans IoT device infrastructures and Cloud/Edge resources (IoT-CANE). The core idea in IoT-CANE is to capture the IoT resource configuration knowledge and then to implement a system that can be updated as and when IoT solutions evolve to facilitate increased acquisition of knowledge. IoT-CANE uses transaction procedures and applies SQL-based

approaches to enable different IoT resource configuration operation recommendations.

The key contributions of this paper are as follows:

- A unified conceptual model capturing resource configurations in IoT environments including those spanning Edge and Cloud.
- A system utilizing our unified conceptual knowledge configuration resource model to recommend valid and appropriate deployment configurations to users.
- An incremental method to facilitate the knowledge acquisition in an IoT resource configuration knowledge base to ensure relevance of IoT-CANE is maintained to reflect IoT and Cloud/Edge innovations.
- A service pipeline for converting context information captured from user requirements into optimal IoT resource configurations solutions.

The rest of this paper is organized as follows. Related work is presented in Section 2. A detailed description of the conceptual model and system architecture embodied within IoT-CANE is presented in Section 3. Our rule based approach to recommendation techniques employed within IoT-CANE are presented in Section 4 with detailed descriptions of design and implementation presented in Section 5. Evaluation of a use case study is provided in Section 6 with conclusions and future work described in Section 7.

2. Related Work

In this section we consider configuration management, conceptual models, and configuration recommendation systems suitable for IoT application deployment across IoT and Cloud/Edge computing. We highlight how research challenges have generated significant work in the different areas of IoT and Cloud/Edge computing, but gaps in requirements still suggests a strong case for our work on IoT-CANE.

2.1. Multi-layer Resources Configuration Management Issues in IoT

The recent trend in composing Cloud applications is driven by connecting heterogeneous services deployed across multiple datacenters [11] [12]. Such a distributed deployment aids in improving IoT application reliability and performance within Cloud/Edge computing environments. Ensuring high levels

of dependability for IoT data transformation tasks composed by a multitude of systems is a considerable issue to tackle from a deployment perspective. In an IoT environment a significant technical challenge is presented when several small services adopted by smart devices and Edge/Cloud infrastructures need to be aggregated in order to produce a new service [13]. However, this service composition problem is only a subset of a number of IoT resource configuration management challenges because configuration management issues for example are also considered in the reusability and optimization perspective within Cloud/Edge infrastructures. Different frameworks for describing and deploying Cloud/Edge resources are proposed in academia and industry. Multiple Cloud providers such as CA AppLogic [14] and AWS OpsWorks [15] allow description and deployment of a complete Cloud application stack. They offer resource representations that are specific to a particular provider. In Edge computing, Docker [16] provides deployment and configuration management solutions for Edge devices based on container techniques [17]. However, when it comes to IoT, in addition to the Cloud and Edge layers, IoT resource configuration management should also consider the physical devices which are deployed widely in most IoT applications. In IoT, all resources from multiple layers need to be considered in a single application which leads to greater complication in ensuring that configurations of IoT device infrastructures coupled with those of propriety Cloud/Edge deployments are correct.

2.2. Conceptual Models in IoT

The Semantic Sensor Networks ontology presents a high-level conceptual model to describe physical devices, their capabilities and the associated properties in the semantic sensor networks within IoT [18]. Authors in [19] provide a description of IoT applications and a data model capturing the relationships across various data providers. They also illustrate how models can be associated with each other and to different application domains. The IoT-A project describes services, entities and resources as basic concepts in IoT [20]. IoT services reveal functionality of a resource hosted on a device offering physical access to an entity, which in turn represents the direct interaction of users with software agents. The approach in [21] presents a design of a complete and lightweight semantic description model for knowledge representation in the IoT field. Widely applied rules in knowledge engineering and ontology modeling are considered in their design. However, their model only considers the physical world of the IoT domain and does not consider

Table 1: Comparison of Related Work

Parameter	Related Work							IoT-CANE
	[18]	[19]	[20]	[21]	[22]	[23]	[24]	
IoT device conceptual model	✓	✓	✓	✓	✗	✗	✗	✓
EDC,CDC conceptual model	✗	✗	✗	✗	✗	✓	✓	✓
IoT ontology modelling	✓	✗	✓	✓	✗	✗	✗	✓
knowledge recommendation	✗	✗	✗	✗	✓	✓	✓	✓
incremental knowledge base	✗	✗	✗	✗	✗	✓	✗	✓

Cloud/Edge components, which are important in unifying knowledge representation of IoT applications to ease deployment. This is especially true when considering a complete view of configuration possibilities across an application.

2.3. Context-aware Recommender Systems

The notion of context awareness for the support of service selection and deployment has matured from its initial proposal in [25]. Today, there are different types of contextual information, which are mainly categorized into three classes: physical context, user context, and appliance context. Such classification is based on an adopted perspective (user or application). In addition, the meaning of context is highly dependent on the area domain and structure of the considered application. This has prompted a search for meaningful definitions of the term concept in a variety of application domains. Possible interpretations for applications similar to e-commerce are provided in [26]. Location is a common contextual piece of information, however, it does not necessarily represent the users' geographic location. For example, geographic location awareness is a key issue in ensuring origin of streamed data for on-demand viewing adheres to appropriate copyright laws in a given country (recommending only valid content). In another example, social tag-based joint filtering given in the context of smart TV applications [22] is a context-aware approach where the information of both user and device contexts are considered. In this case, the recommendations are calculated only using users' preferences and the recommendations are re-ranked appropriately. In [23], the authors present a framework to provide declarative context driven knowledge recommendations for federated Cloud resource configuration. In this framework, a recommendation of configuration knowledge of entities based on a given context is provided. In [24], the

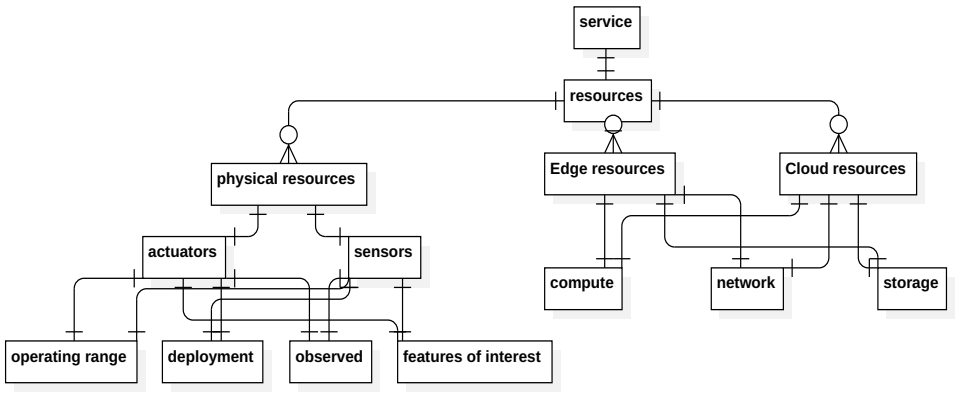

Figure 1: The conceptual model of IoT resources widgets

authors propose an ontology-based infrastructure selection system based on real-time QoS requirements and utilize analytic hierarchy process methods to facilitate multi-criteria decision making for recommendations. Although these frameworks have reached significant reach in terms of real-world deployment and usage, they are limited to Cloud computing and do not address the additional requirements of IoT applications in terms of devices themselves nor their supporting infrastructures.

2.4. Discussion

There is no doubt that significant maturity is demonstrated in the literature for developing solutions to ontology and associated classification concepts, configuration models, and configuration recommendation to ensure suitable evolutionary updating of software/harder. However, the lack of such approaches that span the Cloud/Edge and IoT spectrum of solutions is clear. This has driven our research that has resulted in IoT-CANE. Our contribution in consideration of the work of others is summarized in the comparison table 1.

3. Conceptual Model and System Architecture

In this section we present our approach to Cloud/Edge unified with IoT infrastructure configuration determination and recommendation; the IoT-CANE system architecture.

3.1. Conceptual Model

An IoT framework can take advantage of various models which provide different concepts and abstractions for the components and their respective attributes. The main concepts and abstractions underling a generalized IoT infrastructure and associated relationships are presented in this section. A primary focus of our research is the representation of unified knowledge for IoT resource configuration. A unified hierarchical representation data-model is presented using entity-relationship modeling (ER model), shown in Fig. 1. The physical resources part of our model is based on the Semantic Sensor Network (SSN) Ontology [18]. The SSN ontology is an ontology which depicts various sensors and their observations, related procedures, features of interest, observed attributes, and actuators. The design follows a two-dimensional modularization by implementing a lightweight, but self-contained, core ontology called SOSA (Sensor, Observation, Sample, and Actuator) for its primary classes and attributes. In the SSN ontology, sensors and related concepts are described without domain concepts (e.g., time, locations). In our conceptual model, we are required to consider these domain concepts because our model must consider suitability of the application context to achieve a unified approach for configuration recommendation. Moreover, Cloud/Edge resources are considered in addition to complete the conceptual infrastructure model for IoT as these resources are also configurable and describable and provide a holistic solution satisfying all IoT application configuration requirements.

In Fig. 1, the widget labeled resources consists of the three main entities described as physical resources, Edge resources, and Cloud resources. These three entities represent the abstraction of physical devices (e.g., sensors, actuators), Edge devices (e.g., gateways, routers) and Cloud infrastructures (e.g., Datacenters, servers). In general, service and resource represent the main concepts in IoT applications. Any IoT service must access IoT resources in order to satisfy user requirements. Physical resources are used to capture physical data from sensors and process commands on actuators; Edge resources and Cloud resources conduct all related processing in IoT applications. We now discuss these entities in detail.

- *service*: the service entity represents the domain information of IoT applications, each providing an organized and standardized interface that offers all the necessary functionality provisioning interaction. As such, a service exposes functionality via the way resources may be accessed. The service entity consists of a service profile, service grounding, and

service model subclasses. The type of service can be categorized via Web Ontology Language for Service (OWL-S), Unified Service Description Language (USDL), Web Service Modelling Language (WSML), Web Application Description Language (WADL), Simple Object Access Protocol (SOAP). A service profile describes the semantic description and textual information of a service. Attributes for service interaction and access such as endpoint addresses, input and outputs are provided by the service grounding with the service model depicting the operations and outcomes belonging to a service.

- *resource*: describes available resources across physical device and Cloud/Edge resources, containing identification information regarding each resource (e.g., name, type, access interface, description).

- *physical resource*: describes the attributes of sensors and actuators including their type, location name, latitude, longitude, and availability. The type of sensors and actuators are abstracted into the three categories of physical (e.g., temperature sensor), virtual (e.g., Facebook), and smart (e.g., smart camera). The smart here indicates the physical resource which has the computational capacity to be programmed, enabling actions exhibiting a degree of awareness and autonomy.

- *operating rage*: indicates the conditions a sensor/actuator is expected to operate within and contains such information as resolution, response time, measurement range, precision, latency and accuracy.

- *deployment*: describes the deployment of sensor/actuator for a defined purpose. For instance, a motion sensor can be deployed on the corner of a room to detect entry.

- *observed*: indicates a sensor/actuator that is observed in a particular method allowing the linking of sensor/actuator to feature of interest and consists of accuracy, observation, observed result, result time and sampling expectation.

- *feature of interest*: describes an object associated with a sensor/actuator and associated observation. For example, when you measure the depth of a river then 50 meters may be the observation result and the river is the feature of interest.

- *Edge resource*: indicates the resources deploying in the Edge to leverage computational capacity to improve such items as latency, privacy and security. May describe many attributes including performance, location name, availability, resource type, and attached IoT device to enable sending captured data to actuator.

- *Cloud resource*: describes the Cloud infrastructure deployed in a Cloud service provider. This may include performance, location name, geolocation, availability and associated edge devices of a variety of cloud resources such as virtual machine, container, storage. Both Edge resource and Cloud resource have sub-classes compute, network and storage to enable categorization of descriptions.

- *compute*: depicts the computational capacity of each Cloud/Edge resources. Hypervisor, CPU number, CPU cores, RAM, and operating system are the main attributes of the compute class. In an IoT application, resource availability may influence the decision of resource configuration selection. For example, if deploying a Hadoop cluster to process large volume data in a high performance virtual machine may be a better choice than choosing a limited computational device such as a raspberry pi.

- *network*: describes the network connections between entities, such as those between Cloud/Edge components as well as the data transformation from sensor to Edge. May include attributes such as response time, network bandwidth, up-link bandwidth, down-link bandwidth, and latency. Manages all entity communications and data transformation tasks which relate to the functionality of stability and fault-tolerance.

- *storage*: provides the storage capacity of Cloud/Edge resources and consists of storage capability, storage type, storage bandwidth and associated performance metrics.

These attributes maintain the functional configuration properties of resources proposed within an IoT application. Based on these properties the deployment and configuration of an IoT application can be eased.

To understand our approach more clearly, we present a partial description for a smart building scenario using Table 2 to identify the component descriptions. Smart buildings that provide managed energy efficiency, ease

Table 2: Partial infrastructure components description and instance of smart building in IoT data model

Class	Attribute	Type	Description	Instance in Smart Building
resource	*hasName*	*String*	*Name of the resource*	*Smart temperature sensor*
	resourceType	*Resource type*	*Resource type, such as physical resource, edge resource and cloud resource*	*Physical resource*
	accessInterface	*Interface type*	*Interface type, such as REST, SOAP and XML-RPC*	*RESTful API*
	hasDescription	*String*	*Description of the resource*	*Intelligently measure the temperature of the specific area*
	hasTag	*String*	*Any tags of the resource*	*Smart temperature*
actuator	*actuatorType*	*Actuator type*	*Actuator type, can be physical(locker), virtual(social media) and smart*	*physical*
	locationName	*String*	*Geographical area in which the resource is located*	*Newcastle*
	latitude	*Float*	*Latitude of the actuator*	*54.9783 N*
	longitude	*Float*	*Longitude of the actuator*	*1.6178 W*
	availability	*Boolean*	*Actuator availability*	*available*
sensor	*sensorType*	*Sensor type*	*Sensor type, such as physical(temperature), virtual(social media) and smart*	*physical*
	locationName	*String*	*Geographical area in which the resource is located*	*Newcastle*
	latitude	*Float*	*Latitude of the sensor*	*54.9783 N*
	longitude	*Float*	*Longitude of the sensor*	*1.6178 W*
	availability	*Boolean*	*Sensor availability*	*available*
Operating range	*resolution*	*String*	*The smallest difference in the value of an observable property being observed that would result in perceptible different values of observation results*	*The smallest difference in the value of an observable property being observed that would result in perceptible different values of observation results*
	responseTime	*String*	*The time between a change in the value of an observed and a sensor*	*1.8-60 seconds*
	MeasurementRange	*String*	*A set of values that the sensor can return as the result of an observation*	*-55 to 150,C*
	Precision	*String*	*As a sensor: the closeness of agreement between replicated observations on an unchanged or similar quality value; As an actuator: the closeness of agreement between replicated actuations of an unchanged or similar command*	*0.36 C (max)*
	Latency	*String*	*The time between a command for an observation and the sensor providing a result*	*500ms (max)*
	accuracy	*String*	*The closeness of agreement between the result of an observation (command of an actuation) and the true value of the observed*	*0.1 C*

of accessibility, and automated security is a popular demonstrator for IoT and therefore should be readily understandable in the context of IoT-CANE. A particularly important aspect of such a demonstrator is video surveillance and human behavior analysis in a scene [27, 28, 29]. Such research describes motion detection techniques and its usage in a number of smart building scenarios indicating the utilization of Cloud/Edge for analysis (e.g., longitudinal and streamed) together with sensors (e.g., cameras, motion).

3.2. System Architecture

Our approach automates a configuration knowledge artifact (CKA) suggestion based on user requirements within IoT resource configuration management during such activities as deployment and parameter modification. As shown in Fig. 2, recommended suggestions are generated based on a users' context information. This context information represents an individualized IoT data transformation task or an IoT service requirement. Four context information categories (service category, data source, programming model and deployment node) are chosen to model user context information of a particu-

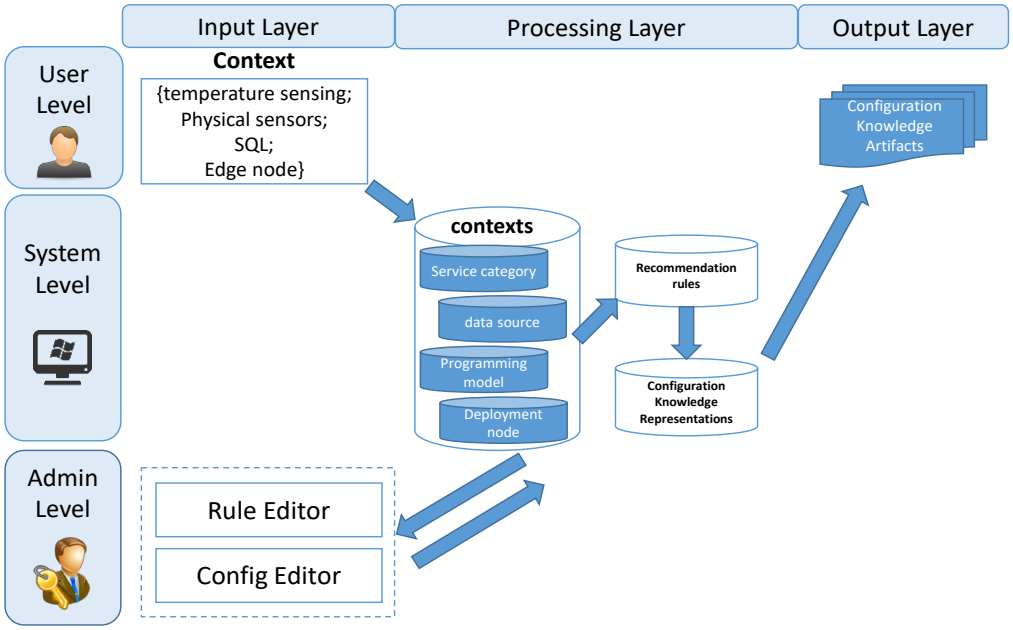

Figure 2: The system architecture of IoT-CANE

lar IoT application in various deployment environments. "Service category" provides the classification of IoT services which can be reused in multiple IoT applications. For example, sensing can be a common IoT service category adopted in different scenarios, such as smart homes (temperature sensing) and smart traffic (motion sensing). "Data source" indicates the origins of the data which can be physical (e.g., sensors) and virtual (e.g., social media). "Programming model" refers to the logical execution processes and data management approaches of a user's IoT application, such as stream processing and batch processing. "Deployment node" represents the deployment place (such as mobile phone or cloud datacenter) of the specific service or data transformation task.

All the necessary instructions and information required to satisfy the context description for resource configuration of Cloud/Edge is included in a recommended configuration knowledge representation (CKR). Recommendations can be derived via CKAs (e.g., bundled virtual appliances and runnable deployment texts) using information from similar past contexts. Recommended CKAs can be accepted unchanged or modified according to user requirements. Users may generate a new CKA if they refuse a recommendation

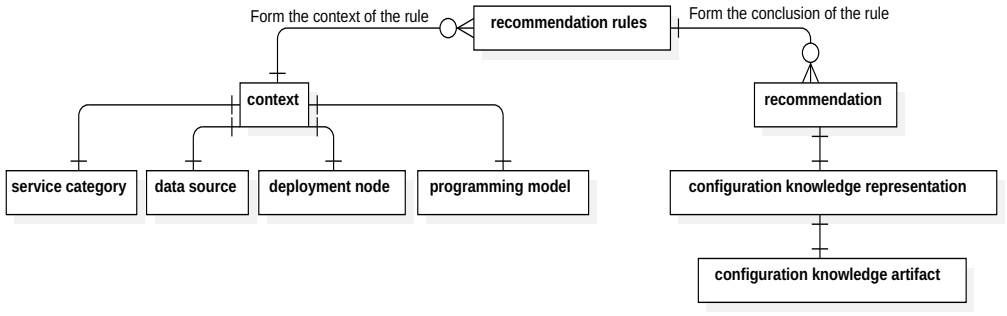

Figure 3: ER diagram of recommendation rules

(under administrative guidance). After any new CKA defined by the user, the system converts these new modifications into recommendation rules and saves them to ensure availability for future recommendations. Meanwhile, recommended CKAs are input into a Docker deployment engine to provide detailed configurations. The detailed description of IoT-CANE will be presented in section 5.

4. Recommendation System Technique

IoT-CANE adopts a rule-based method to generate context-aware configuration recommendations. Ripple Down Rules are employed to facilitate knowledge acquisition. Detailed techniques are discussed in the section given below.

4.1. Recommendation Rule

In IoT-CANE an IoT resource configuration knowledge base (CKB) is maintained to store contextual information regarding CKRs and CKAs. An association is maintained between the items in the CKB by the recommendation rules as shown in Fig. 3. Recommendations include two components, named contexts and conclusions.

Contexts. The left-hand side of the ER diagram contains the context information maintaining "contexts" data of the intended service category (e.g., temperature sensing, motion sensing), data source (e.g., physical sensors, social media APIs), programming model (e.g. streaming process, batch process, SQL, NoSQL) and deployment node (e.g. Edge node, Cloud node). The

CKB is intended to capture metadata and common information about classes having comparable application and resource requirements. Shared context knowledge is allowed to be customized and reused by IoT users. Coordinating contexts with the CKRs can split these representations on the bases of satisfying services effectively.

Conclusions. The components that form the conclusion of the generated recommendation rule is represented on the right-hand side of the ER diagram. CKR is suggested by the generated recommendation rules. The CKR can be deployed by the user via a defined configuration deployment engine such as Docker. Sometimes, users may be required to submit knowledge representations to particular deployment engines and create some CKAs. For example, generating an appropriate image using knowledge representation allowing the user to upload the image to Docker for deploying onto Cloud/Edge services. The deployed resources configuration can be managed by users at any time.

4.2. Single Conclusion Ripple Down Rules

To model heterogeneous IoT resource configurations and to facilitate adequate reuse of existing CKRs, we employ the commonly use knowledge acquisition and maintenance approach of Ripple Down Rules (RDR) [30]. RDR are a form of knowledge acquisition technique that extract knowledge from human experts by grounding that knowledge in terms of the context in which the expert applies or uses that knowledge. In the RDR framework, the knowledge of experts is acquired in order to increase the domain knowledge base. The decision to choose RDR enables the re-usability of the existing CKRs and CKAs. This also enriches the CKB by creating and attaching new rules for later use. Many domains (e.g., database cleansing, UI artifact reuse, NLP) have successfully implemented the RDR technique. Based on our knowledge, RDR has not been adopted in the IoT resource configuration representation field.

Single conclusion RDR, multiple classification RDR and collaborative RDR are the common variations of RDR available [31]. IoT-CANE utilizes a single conclusion RDR technique that considers only one conclusion for given contextual information (to ensure no ambiguity).

5. Design and Implementation

In this section, we present the overall system design including the system workflow and the recommendation rule tree of IoT-CANE.

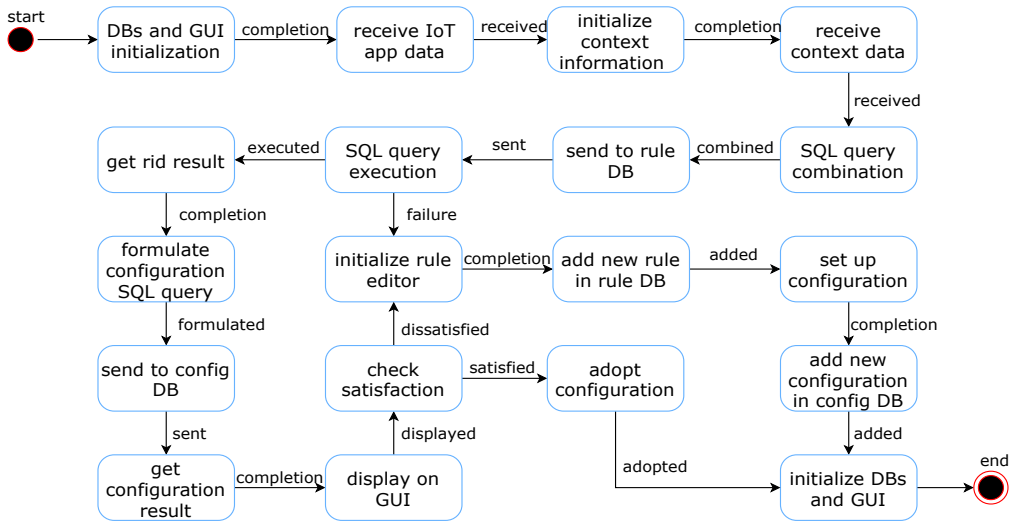

Figure 4: State transition diagram of IoT-CANE

5.1. System Design

Fig 4 depicts a state transition diagram of IoT-CANE which includes various states from start to finish. A design condition must be accomplished before moving the system from one state to another state in the direction of the arrow.

In order to get the appropriate recommendations of resource configuration we employ knowledge from our conceptual model to specify each property in diverse IoT resources (physical resources, Edge resources and Cloud resources). Each property will be set in the config editor module of IoT-CANE based on expert experience (derived from experts in the domain of interest) to make sure these configurations are appropriate. These configurations are stored in the config database (DB). To ensure relevance in dynamic IoT environments, these configurations can be optimized and matured based on user feedback when using IoT-CANE. However, only an administrator can operate the config DB and associated rules governing configuration possibilities to ensure correctness, stability and consistency. Such operations can be adding rule combinations; deleting rule combinations; modifying rule combinations; changing association between rules.

Because each resource configuration combines a large set of attributes, IoT-CANE assumes responsibility for the choices associated to attributes to

compensate for lack of user knowledge. The aim is to avoid user confusion via the use of context information categories (service; data source; programming model; deployment node). These categories abstract the resource configurations in the IoT applications to ease the burden on the user:

- *Service Category*: for each IoT application, we abstract a list of services to indicate the currently available services in the specific IoT application. This list may be updated based on both experts' experience and users' feedback. The demand of Cloud/Edge resources in diverse IoT applications are significantly different, that is why the choice of service category has the most significant influence over the CKR, and the reason why this is the first, primary, context information category.

- *Data Source*: identifies where original raw data comes from. These can be geographically distributed and the raw data can be type non-sensitive. For example, a smart building application could have the same temperature sensors deployed on different floors. A temperature sensor which captures temperature measurements, and a camera which captures image data can be considered different types of raw data. Different CKRs will be recommended for each situation concerning varying different types of data sources and distributed geographical data sources.

- *Programming Model*: widely used execution approaches include: stream (e.g., Kafka Streams), batch (e.g., Hadoop), SQL (e.g., MySQL), NoSQL (e.g., MongoDB). For each programming model, the recommended CKRs should be different to ensure alignment of available properties exhibited by each model in the context of user requirements. For example, the streaming process may require a higher rate of service availability and greater bandwidth of network than batch processing as streaming may require rolling window approaches for uninterrupted flows.

- *Deployment Node*: depicts the physical or virtual node available for deployment. In IoT applications, such deployment nodes can be categorized into Cloud and Edge. However, these categories can be specified in a more detailed manner including gateway, raspberry pi, mobile phone, for the Edge and platform (e.g., Amazon EC2, Microsoft Azure) for the Cloud. The CKRs for Edge nodes and Cloud nodes are significantly different due to their inherent configuration possibilities. For

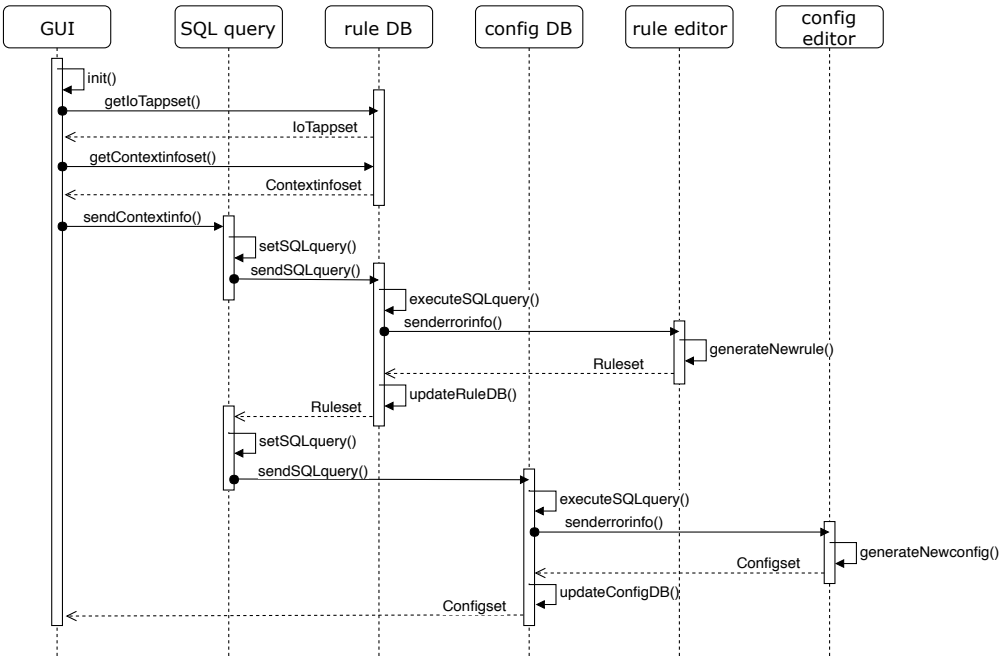

Figure 5: Sequence diagram of IoT-CANE

example, the configuration in a gateway may include DNS and ipv4 address settings, but these configurations are not available in Amazon EC2. In addition, differences across instance types within the two categories may also be different. For example, Amazon EC2 and Microsoft Azure still present different configuration possibilities.

After capturing this context information, a relatively unique CKR can be recommended from IoT-CANE to cover a user's requirement to enable suitable IoT application deployment.

5.2. System Workflow

Fig. 5 shows the system workflow of IoT-CANE. First, the graphical user interface (GUI) initializes by retrieving the IoT application set from the rule DB. This is followed by indication of choice with respect to IoT application available, followed by a get method issued to the rule DB to get the context information for updating context options presented to the user. The user will specify a context including service category (sc), data source

(ds), programming model (pm) and deployment node (dn). This is then sent to the SQL query module to allow the construction of an SQL query to run in the rule DB in order to index appropriate rules required by the user to fulfill their IoT application requirements. In the worst case nothing is returned. This would raise an error message with context information responsible for the error sent to the rule editor module. After intervention to ensure a new rule set can be generated based on the context information of concern, a transfer to the rule DB occurs to ensure a suitable updating of the index. An SQL query is then composed based on the returned rule set in order to search relevant IoT resource configurations. An error message including the chosen rule may be sent to the config editor module in the worst case scenario if, again, user requirements may not be satisfied. Given this eventuality, a specific IoT resource configuration set associated with the sent rule will be generated based on the administrator's experience and expertise together with the given context information for future association. After generation, the config DB module will receive the new resource configuration and an update operation will run automatically. This result will be displayed on the GUI and the user is prompted to evaluate the returned resource configuration.

The request for resource configuration representation recommendation in IoT-CANE is expressed as SQL queries. Now we explain the steps which are executed for resolving a resource configuration representation recommendation request.

- System combines user's input context information to a temporary SQL query;
- The temporary SQL query will be executed in the recommendation rule database to produce a possible result;
- Based on the result, a map of the result to the configuration representation database together with rule number is shown in the GUI;
- Depending on user satisfaction, a new rule and configuration representation will be adopted in the respective database after any required administrative updates.

We now use the smart home example to further the descriptions of IoT-CANE in a real-world context.

A householder purchases a new smart camera to deploy an intrusion detection service in their smart home application. They may not understand the options regarding the addition of the new smart camera within the smart home application in terms of reconfiguration of existing IoT components: which configurations will change and how to change them? IoT-CANE would allow the householder to input the necessary contextual information under a degree of guidance. In this case, they need to choose 'IoT application' as 'Smart Home', 'Service Category' as 'Intrusion Detection', 'Data Source' as 'smart camera', 'Programming Model' as 'Streaming' and 'Deployment Node' as 'Gateway' respectively. IoT-CANE may then combine selected contextual information to a temporary SQL query for execution in the CKB. If there is an existing combination of contextual information in the CKB that matches those selected then the CKR of an intrusion detection service deployed with the appropriate smart camera gateway will be displayed. However, two unexpected situations can impact these results. The householder may not be satisfied with the provided CKR, which means the feedback from user is "not satisfied". This situation will be met differently depending on a number of factors (e.g., householder has own knowledge with regards to appropriate CKR and knows exactly what they need; home owner is not skilled in IoT CKR and can't deploy their application based on the provided CKR, CKR does not satisfy their requirements). Another situation is the temporary SQL query cannot acquire the corresponding result (there is no associated CKR to the request provided, as encoded into the SQL query by the householder). Within this situation, the system will go to another layer to process the SQL request: the administrator may add additional information for the new rule in the CKB with provided contextual information in the rule editor module. This will require adding a corresponding CKR using the administrators expertise and experience in the config editor module. After updating the CKB, the householder can review the returned resource configuration set formulated in JSON format on the IoT-CANE GUI. The example CKR in this scenario is shown in Fig 6.

After the CKR is recommended by IoT-CANE, other resource orchestration engines may be considered to enforce the desired IoT configuration (particularly in the Cloud/Edge domain where such work is mature) to optimize the performance or cost in the IoT application. A set of deployment methods (e.g., Docker deployment) can be adopted using the recommended CKR.

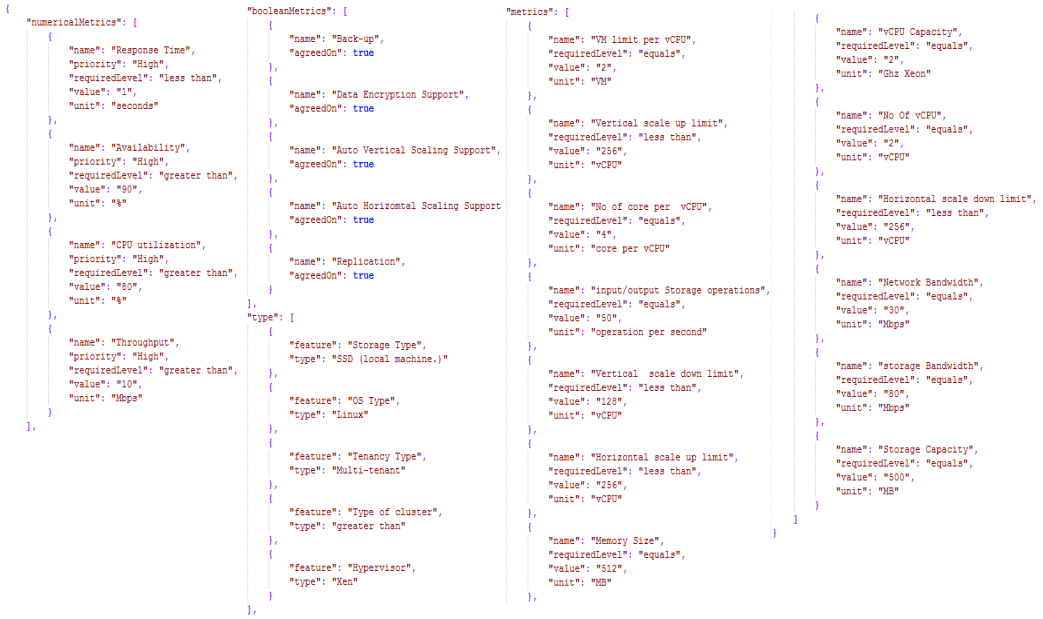

Figure 6: Example of Configuration Knowledge Representation

5.3. Recommendation Rule Tree

In IoT-CANE a tree architecture is used to organize connections across recommendation rules. Fig. 7 depicts the tree representation of the recommendation rules in a CKB. A default conclusion labeled "unknown" is contained in Rule A0. This conclusion is suggested when there is no specified input context; meaning that the service category, data source and other information is not provided in the input context. There are two possibilities available within the structure consisting of "if not" (false) branches and "except" (true) branches to choose from. When IoT-CANE receives a new user context the system parses the rule tree starting from root by checking the node is "if not" or "except", by comparing with the recommendation rules. This step is repeated until the branch search is exhausted. The final conclusion received from the last "except" node will be provided to a user. A similar procedure is performed for each parameter (such as service category, data source, programming model and deployment node) to ensure a tailored result of sufficient detail to satisfy configuration deployment requirements.

Let us consider an example of an administrator that is required to model an IoT resource configuration for a temperature controlling application as an Edge deployment. We assume that the CKB does not have this service

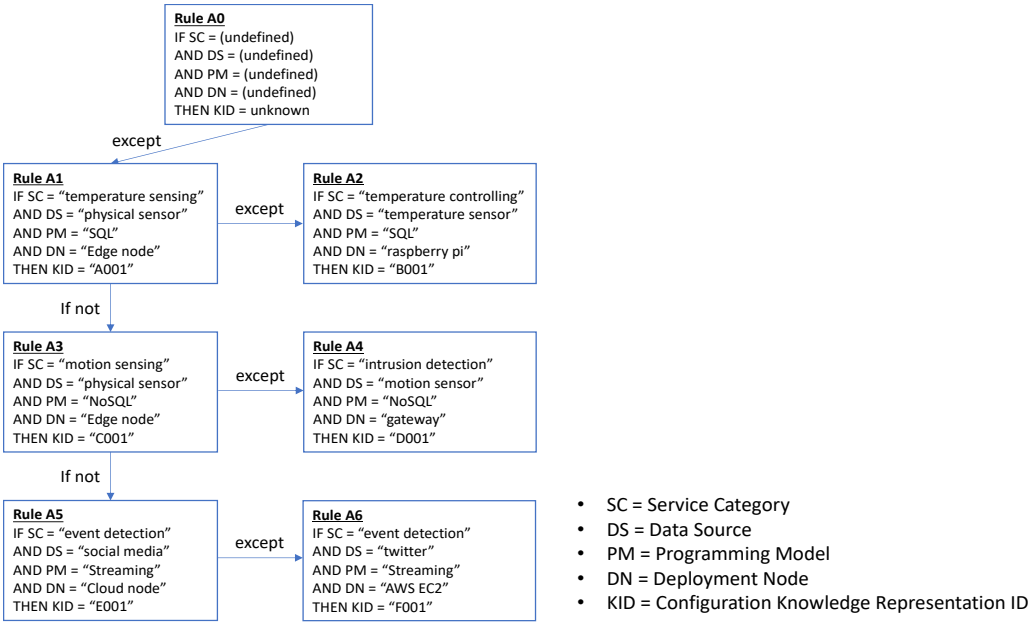

Figure 7: Example of recommendation rule tree structure

definition in any available IoT application. We assume, therefore, there is no Rule A2 in Fig. 7. A query is generated for CKB to find a CKR that is related with service category "temperature sensing" and deployment node "Edge node" (Rule A1). But there is no expert rule available from Rule A1. Therefore, the administrator verifies the CKR linked with Rule A1 and confirms if this CKR is satisfactory for deploying a temperature controlling application. The administrator adds one expert Rule A2 beneath Rule A1 and refers to the modified CKR as the result of Rule A2.

Summarizing, IoT-CANE allows users to focus on the specific infrastructure requirements from the application requirements without requiring users to have knowledge of the technical complexity of multiple IoT resource configuration solutions.

5.4. Computational Complexity

In this section we discuss the computational complexity of the proposed system logic. The model parameters are discussed in detail in Table 3. For the worst case scenario, our proposed system logic considers all the possible combinations (full CROSS JOIN). The total number of options varies based

Table 3: IoT recommender model parameters

Notations	Meaning
Query	A configuration selection query
SC = {sc_1,...,sc_n}	Set of n service categorys
DS = {ds_1,...,ds_m}	Set of m data sources
PM = {pm_1,...,pm_o}	Set of o programming models
DN = {dn_1,...,dn_p}	Set of p deployment nodes
N	Number of rows in the database
M	Number of column in the database

on the number of rows in each table. For the querying process, the upper bound complexity is given in equation (1):

$$O_{query}(sc_n \times ds_m \times pm_o \times dn_p) \tag{1}$$

However, modern databases can use different techniques to reduce the computational complexity. For example, HASH JOIN and MERGE JOIN are widely used to reduce the computational complexity. They are $O(M + N)$ and $O(N * Log(N) + M * Log(M))$ respectively.

6. User Evaluation

In this section, we present the experimental setup and user evaluation for IoT-CANE.

6.1. Experiment setup

We host IoT-CANE on a local machine with 64bit Mac OS X operating system. The machine has the following hardware configuration: Processor (2.4 GHz Intel Core i5); Memory (8GB 1600 MHz DDR3); Graphics (Intel Iris 1536 MB); Storage (256 GB SSD). We use MySQL database for our data management requirements.

6.2. User Evaluation

In order to evaluate IoT-CANE we utilize a use case study to investigate performance and acceptance. Ten participants were involved in the experiment. All participants are PhD students working in the Cloud Computing and Internet of Things area at Newcastle University. All of them have experience of deployment and configuration management in Cloud infrastructure

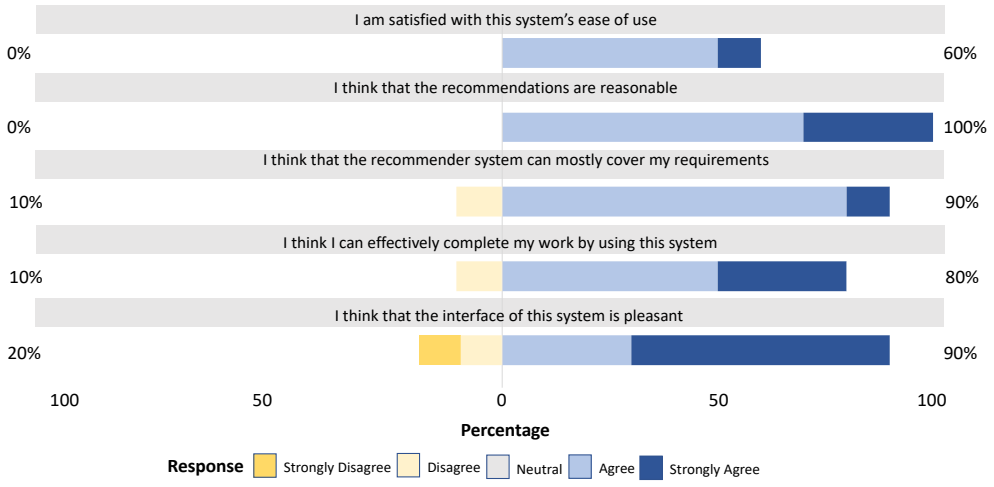

Figure 8: User survey result

and Edge devices. None of them had experience of using an automated IoT resource configuration selection tool.

After using IoT-CANE the subjects were asked to complete a questionnaire. Nine questions are used to evaluate user experience and opinion. The questions are listed below. A selection of the results are shown in Fig. 8.

- How satisfied are you with this system's ease of use?
- How often does our system freeze or crash?
- To what extent do you think that the recommendations are reasonable?
- To what extent does the recommendation system cover your requirements?
- To what extent do you think you can effectively complete your work using this system?
- Do you agree or disagree that the interface of this system is pleasant?
- How likely is it that you would recommend this software to a friend or group member?

- Overall, how satisfied or dissatisfied are you with our recommendations?
- How can we improve our system?

As shown in Fig. 8, most of the participants were satisfied with IoT-CANE in the following ways: ease of use; reasonable recommendations; pleasant user interface. Based on feedback we may conclude, in the most part, that the conceptual model covers the majority of resource configuration knowledge requirements in IoT and IoT-CANE may provide reasonable recommendations to IoT application users. However, not all of the participants were satisfied completely. The final question in the survey asked the participants to give some suggestions to improve our system. Their suggestions can be categorized as follows:

- They wanted new features, such as automatic deployment;
- They suggested that the user interface could be more descriptive and user friendly;
- They suggested that the system could provide multiple choices of the configurations to handle more scenarios.

Based on their suggestions, our IoT-CANE needs to be improved in two ways: (1) provide a new service which converts JSON format configuration files into Docker-readable configuration files (e.g., YAML format); (2) provide an automatic deployment service based on popular container techniques.

7. Conclusion and Future Work

In this paper we present a unified solution to the pressing problem of delivering appropriate IoT solutions in the increasingly complex world of cloud/edge/IoT environments. Todays IoT developers are required to provide gateway technologies across a variety of device manufacturers spanning numerous data analysis techniques supported on distinct Cloud/Edge providers. This is a complex and significant task and provides a large number of ways IoT applications may be configured, deployed and evolved. Considering that todays IoT supporting system infrastructures are dynamic and consistently evolving, the options for determining optimal Cloud/Edge/IoT configurations that satisfy requirements is not straightforward.

In this paper our IoT-CANE solution presents an end-to-end pipeline for classifying, configuring and recommending appropriate solutions in this most complex of environments. Our solution spans IoT device ontology in addition to conceptual models inclusive of Cloud/Edge technologies together with recommender services and an administrative approach to evolving IoT-CANE to ensure continued validity in dynamically changing infrastructures. Furthermore, our approach is based on well-known concepts and languages, making it easily accessible for developers and increasing its applicability in today's environments. We utilize a Ripple Down Rules (RDR) approach for the first time to recommender-based issues within IoT with a single conclusion/classification setting. This not only removes ambiguity, but also affords a well known structured approach to identifying shortcomings in configuration possibilities (allowing administrative updates) while allowing system evolution under expert guidance.

We present a user/developer case study to validate our approach. We show that our main goals are achieved in the context of recommending appropriate solutions and evolving such solutions. In addition, we gain insights into the updates necessary to provision a more accessible solution for end-users.

Our future work will focus on automation of deployment in the context of recommendation systems, bridging the current gap between advice and real-world development. In addition, we will consider a more accessible interface and online deployment utilizing data analysis to provide a learned approach from the patterns of usage given IoT requirements to ensure greater efficiency and flexibility in our approach.

Acknowledgement

This research is supported by the following projects: LANDSLIP: NE/P000681/1, FloodPrep: NE/P017134/1 and PACE: EP/R033293/1.

References

[1] I. Lee, K. Lee, The internet of things (iot): Applications, investments, and challenges for enterprises, Business Horizons 58 (4) (2015) 431–440.

[2] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, Future generation computer systems 29 (7) (2013) 1645–1660.

[3] S. Madakam, R. Ramaswamy, S. Tripathi, Internet of things (iot): A literature review, Journal of Computer and Communications 3 (05) (2015) 164.

[4] Y. Tang, D. Chen, L. Wang, A. Y. Zomaya, J. Chen, H. Liu, Bayesian tensor factorization for multi-way analysis of multi-dimensional eeg, Neurocomputing 318 (2018) 162–174.

[5] D. Chen, Y. Hu, L. Wang, A. Y. Zomaya, X. Li, H-parafac: Hierarchical parallel factor analysis of multidimensional big data, IEEE Transactions on Parallel and Distributed Systems 28 (4) (2017) 1091–1104.

[6] D. Chen, X. Li, L. Wang, S. U. Khan, J. Wang, K. Zeng, C. Cai, Fast and scalable multi-way analysis of massive neural data, IEEE Transactions on Computers 64 (3) (2015) 707–719.

[7] H. Ke, D. Chen, T. Shah, X. Liu, X. Zhang, L. Zhang, X. Li, Cloud-aided online eeg classification system for brain healthcare: A case study of depression evaluation with a lightweight cnn, Software: Practice and Experience.

[8] J. Fan, J. Yan, Y. Ma, L. Wang, Big data integration in remote sensing across a distributed metadata-based spatial infrastructure, Remote Sensing 10 (1) (2017) 7.

[9] M. Blackstock, R. Lea, Iot interoperability: A hub-based approach, in: Internet of Things (IOT), 2014 International Conference on the, IEEE, 2014, pp. 79–84.

[10] ProgrammableWeb, `https://www.programmableweb.com/category/internet-things/api` (2018).

[11] S. K. Datta, R. P. F. Da Costa, C. Bonnet, Resource discovery in internet of things: Current trends and future standardization aspects, in: Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on, IEEE, 2015, pp. 542–547.

[12] A. Alqahtani, Y. Li, P. Patel, E. Solaiman, R. Ranjan, End-to-end service level agreement specification for iot applications, in: 2018 International Conference on High Performance Computing & Simulation (HPCS), IEEE, 2018, pp. 926–935.

[13] K. Dar, A. Taherkordi, R. Rouvoy, F. Eliassen, Adaptable service composition for very-large-scale internet of things systems, in: Proceedings of the 8th Middleware Doctoral Symposium, ACM, 2011, p. 2.

[14] Ca applogic, https://support.ca.com/us/product-information/ca-applogic.html (2014).

[15] Aws opsworks, https://aws.amazon.com/cn/opsworks/ (2018).

[16] Docker, https://www.docker.com/ (2018).

[17] D. N. Jha, S. Garg, P. P. Jayaraman, R. Buyya, Z. Li, R. Ranjan, A holistic evaluation of docker containers for interfering microservices, in: 2018 IEEE International Conference on Services Computing (SCC), IEEE, 2018, pp. 33–40.

[18] W. S. S. N. I. Group, Semantic sensor network ontology, https://www.w3.org/2005/Incubator/ssn/ssnx/ssn (2011).

[19] S. De, P. Barnaghi, M. Bauer, S. Meissner, Service modelling for the internet of things, in: Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on, IEEE, 2011, pp. 949–955.

[20] M. Bauer, N. Bui, J. De Loof, C. Magerkurth, A. Nettsträter, J. Stefa, J. W. Walewski, Iot reference model, in: Enabling Things to Talk, Springer, 2013, pp. 113–162.

[21] W. Wang, S. De, G. Cassar, K. Moessner, Knowledge representation in the internet of things: semantic modelling and its applications, automatika 54 (4) (2013) 388–400.

[22] W.-P. Lee, C. Kaoli, J.-Y. Huang, A smart tv system with body-gesture control, tag-based rating and context-aware recommendation, Knowledge-Based Systems 56 (2014) 167–178.

[23] D. Weerasiri, B. Benatallah, Unified representation and reuse of federated cloud resources configuration knowledge, in: Enterprise Distributed Object Computing Conference (EDOC), 2015 IEEE 19th International, IEEE, 2015, pp. 142–150.

[24] M. Zhang, R. Ranjan, M. Menzel, S. Nepal, P. Strazdins, W. Jie, L. Wang, An infrastructure service recommendation system for cloud applications with real-timeqosrequirement constraints, IEEE Systems Journal.

[25] B. Schilit, N. Adams, R. Want, Context-aware computing applications, in: Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on, IEEE, 1994, pp. 85–90.

[26] Z.-L. Chen, S. Raghavan, P. Gray, H. J. Greenberg, State-of-the-art decision-making tools in the information-intensive age (2008).

[27] W. Lao, J. Han, P. H. De With, Automatic video-based human motion analyzer for consumer surveillance system, IEEE Transactions on Consumer Electronics 55 (2) (2009) 591–598.

[28] X. Zhang, Q. Yu, H. Yu, Physics inspired methods for crowd video surveillance and analysis: a survey, IEEE Access 6 (2018) 66816–66830.

[29] T. S. A. Arias, S. D. R. Castillo, C. C. E. Martínez, P. L. M. Aguirre, P. M. A. Pico, N. H. W. Bano, F. R. N. Cobo, Human behavior ontologies: Surveillance task, in: 2018 13th Iberian Conference on Information Systems and Technologies (CISTI), IEEE, 2018, pp. 1–9.

[30] B. R. Gaines, P. Compton, Induction of ripple-down rules applied to modeling large databases, Journal of Intelligent Information Systems 5 (3) (1995) 211–228.

[31] B. Kang, P. Compton, P. Preston, Multiple classification ripple down rules: evaluation and possibilities, in: Proceedings 9th Banff knowledge acquisition for knowledge based systems workshop, Vol. 1, 1995, pp. 17–1.