Ontology and Reasoning on Device Connectivity

Evelina Pencheva[1], Ivaylo Atanasov[1], Anastas Nikolov[1], Rozalina Dimova[2]

Abstract – **In this paper, we present an approach to cloud application orchestration. The approach allows detection and resolution of interactions among cloud applications. The approach is based on ontology for IoT device connectivity. It is illustrated for applications which add functionality to basic bearer selection procedure.**

Keywords – **Connectivity management, Cloud orchestration, Diagnostic and monitoring traps, Ontology, Reasoning.**

I. Introduction

The amount of data generated by connected device increases exponentially with the ubiquitous penetration of Internet of Things (IoT). Cloud computing is a way to alleviate the data problem. It involves delivering data, applications, multimedia and more over the Internet to data centers. Both technologies serve to increase efficiency in different application areas.

Device-to-cloud communication involves an IoT device connecting directly to an Internet cloud application to exchange data and control message traffic. It often uses wireless connections, but can also use cellular technology [1], [2]. Different technologies have different requirements for quality of service (QoS), which complicates the logic for bearer selection. Furthermore, the logic for bearer selection may be based on different policies such as the device location and the requirements for charging.

Cloud connectivity lets an application to obtain remote access to a device. It also potentially supports pushing software updates to the device.

Cloud orchestration is programming that manages the interconnection and interactions among cloud-based applications. To orchestrate cloud applications is to arrange them so they achieve a desired result. A comparative study on existing approached to cloud service orchestration is presented in [3]. In [4], the authors present layer architecture for cloud service orchestration. A decentralized approach to the orchestration of cloud services using multi-agent system is proposed in [5]. In [6], the authors present an autonomic framework for cloud computing orchestration based on virtual machines migrations and heuristics to select hosts to be activated or deactivated when needed. The survey on research related to cloud orchestration shows that works deal with high level architectural aspects and do not provide more details on detecting and resolving of interactions among applications.

In this paper we propose an approach to cloud application orchestration. The approach allows detection and resolution of interactions among cloud applications. It is illustrated for applications which add functionality to basic bearer selection procedure for IoT devices.

The paper is structured as follows. In Section II, a semantic data related to IoT connectivity management is presented. Section III describes cloud applications which manage device connectivity based on different policies. Possible service interactions and their resolution are discussed in Section IV. The conclusion discusses implementation aspects of the proposed method for service orchestration and highlights its benefits.

[1]The authors are with the Faculty of Telecommunications at Technical University of Sofia, 8 Kl. Ohridski Blvd, Sofia 1000, Bulgaria, E-mail: {enp, iia}@tu-sofia.bg

[2]R. Dimova is with the Faculty of Computing and Automatics at Technical University of Varna, Studentska str, Varna, Bulgaria, E-mail: rdom@abv.bg

II. Device Connectivity Management Ontology

Our research is based on Open Mobile Alliance (OMA) trap mechanism which may be employed by an application to enable the device to capture and report events and other relevant information generated from various components of the device, such as a protocol stack, device drivers, or applications [7]. OMA traps that may be used for connectivity management are geographic trap, received power trap, call drop trap, quality of service (QoS) trap, and data speed trap [8].

In order to send information over the network, any IoT device needs connectivity. A connected device uses a network bearer and can measure its signal strength. A possible sequence of procedures performed by the server running cloud application for connectivity management and wireless device in the context of connectivity management is as follows. The server establishes an observation relationship with the device to acquire periodical or triggered notifications about signal strength of the used bearer. The device sends periodical or triggered notifications about signal strength. Upon dropping of signal strength under application defined threshold, the server queries about used and available network bearers. In case the device senses available unused bearers, the cloud application may initiate bearer selection. Different cloud applications may use different policies for the bearer selection. For example, a cloud application may apply location based policy for bearer selection, while another cloud application may initiate bearer selection procedure whenever an uplink or downlink average data speed reaches an application defined lower or higher threshold value. Fig. 1 shows the ontology related to connectivity management of IoT devices.

In the figure, a bearer change is required for the device when it experiences bad signal whose signal strength is under application defined value. In addition to basic concepts and properties related to basic connectivity management, the figure show concepts and properties related to location based and data speed-based bearer selection. A cloud application may define geographic area in which a preferred bearer has to be used. Another cloud application may define thresholds indicating low and high uplink and downlink speeds, and when the data speeds are below/above low/high thresholds the application considers the speed as unacceptable.

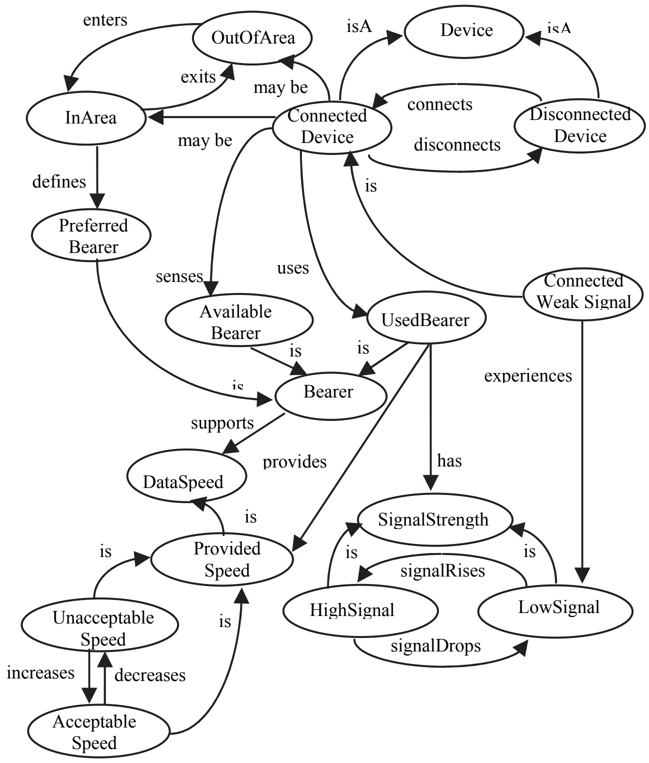

Fig. 1. Ontology for connectivity management of IoT devices

The proposed ontology may be described by OWL. For sake of brevity we describe the ontology by description logic.

The following concepts express the device state and facts related to the device connectivity:

- disconnected – the device is disconnected;
- $connected_b$ – the device is connected by bearer *b*;
- $weakSignal_b$– the device is connected by bearer *b,* but the signal strength of *b* is low;
- $available_b$ – the bearer *b* is available.
- Roles represent actions or notifications about events related to device connectivity management.
- connects – device connects to the network;
- disconnects – the device disconnects from the network;
- change – the device changes the used bearer;
- signalDrops – the signal of the used bearer drops;
- getParameters - the server queries the device about connectivity parameters;
- parameters - the device provides the requested connectivity parameters;
- changeBearer - the server instructs the device to change the used bearer.

Concepts and roles are used to specify the connectivity management model (CMM). The Terminology box (TBox) consists of expressions that represent how the device can change its state.

$$disconnected \sqsubseteq \exists connects.connected_b \quad (1)$$

$$connected_b \sqsubseteq \exists getParameters.connected_b \quad (2)$$

$$connected_b \sqsubseteq \exists parameters.connected_b \sqcap available_c \quad (3)$$

$$connected_b \sqsubseteq \exists parameters.connected_b \sqcap \neg available_c \quad (4)$$

$$connected_b \sqsubseteq \exists (signalDrops \sqcap getParameters).weakSignal_b \quad (5)$$

$$weakSignal_b \sqsubseteq \exists parameters.(weakSignal_b \sqcap available_c) \quad (6)$$

$$weakSignal_b \sqcap available_c \sqsubseteq \exists changeBearer.connected_c \quad (7)$$

$$weakSignal_b \sqsubseteq \exists parameters.(weakSignal_b \sqcap \neg available_c) \quad (8)$$

$$weakSignal_b \sqcap \neg available_c \sqsubseteq \exists disconnects.disconnected \quad (9)$$

$$connected_b \sqsubseteq \exists disconnects.disconnected \quad (10)$$

$$weakSignal_b \sqsubseteq \exists disconnects.disconnected \quad (11)$$

The Assertion box (ABox) contains one statement presenting the initial state for each device: $s_0{:}\sqcap_{d \in Devices}$ disconnected.

To express the fact that each device is in exactly one state at any moment the following statement is used:

$$\top \sqsubseteq \neg(\sqcup_{d1,d2 \in CMM,\ d1 \neq d2}(s_1 \sqcap s_2)) \sqcap (\sqcup_{d \in CMM}\, s)$$

The device state changes by means of actions defined as action functions. An action function $Func_{CMM}$ for given state corresponds to the possible transitions in the CMM. For example, the expression $Func_{CMS}(connected_b)$= signalDrop} ∪{disconnect} means that, if the device is connected, the received power of the used bearer may drop, the device may disconnect or deregister.

The fact that each device can change the CMM state only by means of certain actions is represented by the following statement: for all $s \in CMM$, and all $R \notin Func_{CMM}(s)$, $s \sqsubseteq \forall R.s$.

III. Adding Functionality to Device Connectivity

A. Location-Based Bearer Selection

The Location-based Bearer Selection (LBS) application assumes that there is a predefined geographic area in which a preferred bearer is used. The state diagram of service logic for location based bearer selection is shown in Fig.2.

Additional concepts representing facts and roles are defined:

- inArea – the device is located in the specified area;
- outOfArea – the device is located out of the specified area;
- $preferred_b$ – the bearer *b* is the preferred one in the specified area;
- enters – the device enters the specified area;
- exits – the device exits the specified area;
- location – the device sends its location;
- getLocation – the server queries about device’s location.

The following trivial axiomis true: $outOfArea \equiv \neg inArea$.

The refinement of the knowledge base for LBS application is defined by the following statements.

When the device is connected, the application queries about device location:

$$LBS \sqcap connected_b \sqsubseteq \exists getLocation.\, connected_b \quad (12)$$

The device responds and the application can determine its location with respect of the predefined geographic area:

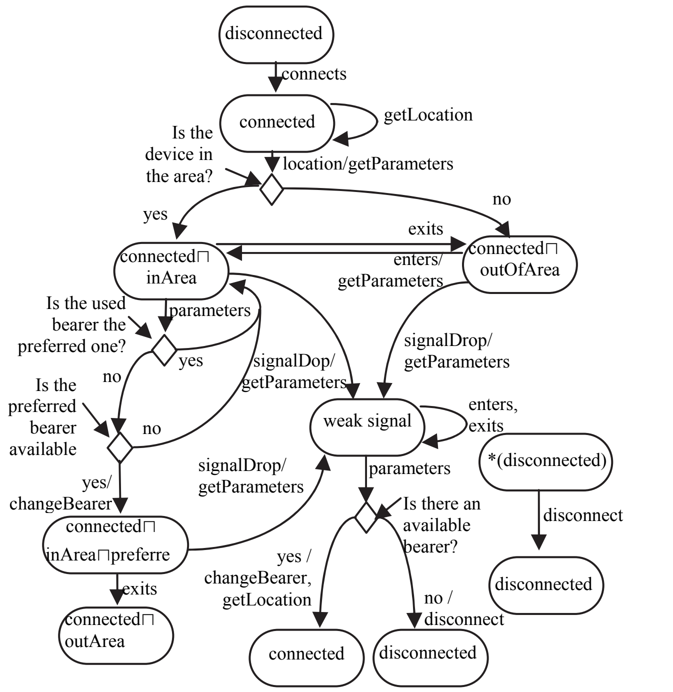

Fig. 2. State diagram of service logic for location-based bearer selection

$$LBS \sqcap connected_b \sqsubseteq \exists location.(connected_b \sqcap inArea) \quad (13)$$

$$LBS \sqcap connected_b \sqsubseteq \exists location.(connected_b \sqcap outOfArea) \quad (14)$$

Based on the device response of the query about connectivity parameters, the application may determine whether the device uses the preferred bearer, or the preferred bearer is among the available ones in case the device is in the area:

$$LBS \sqcap connected_b \sqcap inArea \sqsubseteq \exists parameters.(connected_b \sqcap inArea \sqcap preferred_b) \quad (15)$$

$$LBS \sqcap connected_b \sqcap inArea \sqsubseteq \exists parameters.(connected_b \sqcap inArea \sqcap preferred_c \sqcap available_c) \quad (16)$$

If the device is in the area and the preferred bearer is not used but available, the application initiates bearer change:

$$LBS \sqcap connected_b \sqcap inArea \sqcap preferred_c \sqcap available_c \sqsubseteq \exists changeBearer.(connected_c \sqcap inArea \sqcap preferred_c) \quad (17)$$

The device may enter or exit the area:

$$LBS \sqcap connected_b \sqcap inArea \sqsubseteq \exists exits.(connected_b \sqcap outOfArea) \quad (18)$$

$$LBS \sqcap connected_b \sqcap outOfArea \sqsubseteq \exists enters.(connected_b \sqcap inArea) \quad (19)$$

The application logic is summarized by:

$$LBS \sqsubseteq \neg(connected_b \sqcap inArea \sqcap preferred_c \sqcap available_c) \quad (20)$$

B. Data Speed-Based Bearer Selection

Data speed bearer selection (DBS) application uses the data speed trap. The application configures different data speed traps for uplink and downlink. Low speed data traps become active when the average data speed calculated for the given period reaches below the server defined lower threshold value. High speed data traps become active when the average data speed calculated for the given period reaches above this higher threshold value. The application initiates bearer selection whenever the data speed trap goes to active. The knowledge base for this service is extended with new concepts representing unacceptable for the application data speeds and bearer with appropriate data speed, a new role for trap activity and statements for bearer selection logic:

- $speedUnacceptable_b$ – the data speed is beyond the application defined thresholds;
- dsTrapFires – any of the data speed traps goes active;
- $appropriate_b$ – the data speed supported by bearer *b* are acceptable for the application.

The refinement for DBS service is defined by the following statements.

Being connected to bearer b, the device may experience unacceptable for the application data speeds:

$$DBS \sqcap connected_b \sqsubseteq \exists dsTrapFires.(connected_b \sqcap speedUnacceptable_b) \quad (21)$$

If the data speeds are unacceptable, the application queries the device about its connectivity parameters:

$$DBS \sqcap connected_b \sqcap speedUnacceptable_b \sqsubseteq \exists getParameters.(connected_b \sqcap speedUnacceptable_b) \quad (22)$$

Based on the device response, the application may determine that there is an available bearer which supports acceptable data speeds and the application initiates bearer change:

$$DBS \sqcap connected_b \sqcap speedUnacceptable_b \sqsubseteq \exists parameters.(connected_b \sqcap speedUnacceptable_b \sqcap available_c \sqcap appropriate_c) \quad (23)$$

$$DBS \sqcap connected_b \sqcap speedUnacceptable_b \sqcap available_c \sqcap appropriate_c \sqsubseteq \exists changeBearer.connected_c \quad (24)$$

Based on the device response, the application may determine that there is no available bearer which supports acceptable data speeds and the application initiates device disconnect:

$$DBS \sqcap connected_b \sqcap speedUnacceptable_b \sqsubseteq \exists parameters.(connected_b \sqcap speedUnacceptable_b \sqcap \neg available_c \sqcap appropriate_c) \quad (25)$$

$$DBS \sqcap connected_b \sqcap speedUnacceptable_b \sqcap \neg available_c \sqcap appropriate_c \sqsubseteq \exists disconnects.disconnected \quad (26)$$

The following statement summarizes the application logic:

$$DBS \sqsubseteq \neg(connected_b \sqcap speedUnacceptable_b \sqcap available_c \sqcap appropriate_c). \quad (27)$$

IV. Reasoning on Service Interaction

By the use of OMA Diagnostic and monitoring traps different policies may be used for connectivity management. Further, the bearer selection may depend on available subscriber balance. Real-time information about device provider's balance may be acquired by means of Policy and Charging Control (PCC) functionality. The PCC concept is designed to enable flow based charging including online credit control and policy control which supports service authorization and quality of service management [9].

When introducing new application, it is important to find out whether the new application is contradictory to existing concepts i.e. whether it satisfies or not the statements in the TBox representing the connectivity management model.

Interaction between LBS and DBS occurs when the device is in the specified area and uses the preferred bearer as to LBS, and the data speeds are unacceptable and the DBS requires a change to a bearer which is available one and supports acceptable data speeds. We formally prove our claim.

Proposition 1: Undesired service interaction occurs on activation of LBS⊓DBS.

Proof: Applying standard reasoning to the knowledge base we derive the following sequence of device's state and transitions:

As to (1) s_0 connect s_1: $connected_b$.
As to (12) s_1 getLocation s_1.
As to (13) s_1 location s_2: $connected_b \sqcap inArea$.
As to (2) to s_2 getParameters s_2.
As to (3) s_2 parameters s_3: $connected_b \sqcap inArea \sqcap preferred_b$.
As to (21) s_3 dsTrapFire s_4: $connected_b \sqcap inArea \sqcap preferred_b \sqcap speedUnacceptable_b$.
As to (2) s_4 getParameters s_4.
As to (3) s_4 parameters s_5:$connected_b \sqcap inArea \sqcap preferred_b \sqcap speedUnacceptable_b \sqcap acceptable_c \sqcap available_c$.
As to s_5 changeBearer$_c$ s_6: $connected_c \sqcap inArea \sqcap preferred_b$.
As to (2) s_6 getParameters s_6.
As to (24) s_6 parameters s_7: $connected_c \sqcap inArea \sqcap preferred_b \sqcap available_b$ which contradicts to (20) as to LBS, namely
$\neg(connected_b \sqcap inArea \sqcap preferred_c \sqcap available_c)$■.

The result shows that when applying both applications to the same device, statements representing the LBS and DBS do not satisfy the statements in the knowledge base i.e. both applications contradict to each other.

Once detected, service interactions may be resolved by setting priorities. The cloud functionality for service orchestration determines the required behavior in case of service interaction based on application priority. Application with higher priority can override the instructions of application with lower priority.

Let us denote the priority of *i* service by P_i. Then

$$LBS \sqcap CBS \sqcap P_{LBS} < P_{CBS} \sqcap connected_b \sqcap inArea \sqcap preferred_b \sqcap speedUnacceptable_b \sqcap acceptable_c \sqcap available_c \sqsubseteq \exists changeBearer.connected_b \sqcap inArea \sqcap preferred_c \sqcap available_c \sqcap speedUnacceptable_c \quad (28)$$

V. Conclusion

In this paper we propose an approach to cloud service orchestration. The approach is illustrated for applications which add value to IoT device connectivity management. Each cloud application applies specific policy for the network bearer that has to be used by the device. The approach is based on ontology for device connectivity. The ontology and the application logic may be described by Ontology Web Language (OWL). The service interaction is considered as satisfiability problem and undesired application behavior may be discovered by applying standard reasoning algorithm. There exist a number of ontology editors and frameworks for constructing domain models and knowledge-based applications with ontologies and reasoners to infer logical consequences from a knowledge base.

The proposed method for resolving service interaction using priorities allows dynamic service orchestration.

Acknowledgement

The research is conducted under the grant of project ДН07/10-2016, funded by National Science Fund, Ministry of Education and Science, Bulgaria.

References

[1] J. Zhou, Z. Cao, X. Dong and X. Lin, "Security and privacy in cloud-assisted wireless wearable communications: Challenges, solutions, and future directions," in *IEEE Wireless Communications*, vol. 22, no. 2, pp. 136-144, 2015.

[2] J. Huang *et al.*, "Modeling and Analysis on Access Control for Device-to-Device Communications in Cellular Network: A Network-Calculus-Based Approach," in *IEEE Transactions on Vehicular Technology*, vol. 65, no. 3, pp. 1615-1626, 2016.

[3] K. Bousselmi, Z. Brahmi and M. M. Gammoudi, "Cloud Services Orchestration: A Comparative Study of Existing Approaches," *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, Victoria, BC, 2014, pp. 410-416.

[4] P. Jain, A. Datt, A. Goel and S.C. Gupta, "Cloud service orchestration based architecture of OpenStack Nova and Swift," *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Jaipur, 2016, pp. 2453-2459.

[5] Z. Brahmi and J. Ben Ali, "Cooperative agents-based decentralized framework for cloud services orchestration," *2015 6th International Conference on Information Systems and Economic Intelligence (SIIE)*, Hammamet, 2015, pp. 46-51.

[6] R. Weingärtner, G.B. Bräscher and C.B. Westphall, "A Distributed Autonomic Management Framework for Cloud Computing Orchestration," *2016 IEEE World Congress on Services (SERVICES)*, San Francisco, CA, 2016, pp. 9-17.

[7] Open Mobile Alliance, Diagnostics and Monitoring Management Object. OMA-TS-DiagMonTrapMO-V1_0-20090414-C, 2009

[8] Open Mobile Alliance, Diagnostics and Monitoring Trap Events Specifications,"OMA-TS-DiagMonTrapEvents-V1_2-20131008-A, 2013

[9] 3GPP Technical Specification Group Services and System Aspects; Policy and charging control architecture, Release 13, v13.7.0, 2016.