Test-Enabled Architecture for IoT Service Creation and Provisioning

Suparna De[1], Francois Carrez[1], Eike Reetz[1,2], Ralf Tönjes[2], and Wei Wang[1]

[1] Centre for Communication Systems Research (CCSR), University of Surrey, Guildford, UK
{S.De,F.Carrez,Wei.Wang}@surrey.ac.uk
[2] Faculty of Engineering and Computer Science, University of Applied Sciences Osnabrück, Germany
{E.Reetz,r.toenjes}@hs-osnabrueck.de

Abstract. The information generated from the *Internet of Things* (IoT) potentially enables a better understanding of the physical world for humans and supports creation of ambient intelligence for a wide range of applications in different domains. A semantics-enabled service layer is a promising approach to facilitate seamless access and management of the information from the large, distributed and heterogeneous sources. This paper presents the efforts of the IoT.est project towards developing a framework for service creation and testing in an IoT environment. The architecture design extends the existing IoT reference architecture and enables a test-driven, semantics-based management of the entire service lifecycle. The validation of the architecture is shown though a dynamic test case generation and execution scenario.

Keywords: Internet of Things, Architecture, Automated Test Derivation, Semantic IoT Services.

1 Introduction

A dynamic *Service Creation Environment* (SCE) that gathers and exploits data and information from the heterogeneous sources can help to overcome the technological boundaries and dynamically design and integrate new services and business opportunities for the *Internet of Things* (IoT). Rapid service creation and deployment in IoT environments requires a large number of resources and automated interpretation of environmental and context information. Moreover, the mobility of resources and the dynamicity of the environment necessitate integration of test-friendly description capabilities in the development and maintenance of services from the beginning. Integrating service oriented computing mechanisms and semantic technologies to create a semantic service layer on the top of IoT is a promising approach for facilitating seamless access and management of the information from the large, distributed and heterogeneous sources. The application of semantics to enable automated testing of the IoT services can reduce the time to deployment, while context-aware service adaptation can support deployed services to respond to environmental changes. In this paper, we present an architecture for service creation and testing in an IoT environment, which

A. Galis and A. Gavras (Eds.): FIA 2013, LNCS 7858, pp. 233–245, 2013.

has been developed as part of the *IoT Environment for Service Creation and Testing* (IoT.est)[1] EU ICT-FP7 project.

The IoT-Architecture (IoT-A)[2] EU ICT-FP7 project has proposed an *Architectural Reference Model* (ARM) [1] for the IoT, which identifies the basic IoT domain concepts and the functional components of a *Reference Architecture* (RA). Our contribution is the adoption and application of these principles in order to develop a coherent architecture for self-management and testing of IoT services. This is illustrated by mapping the identified functionalities within IoT.est into the IoT-A ARM, while also including extensions for IoT-enabled testing. Our second contribution is the implementation of the architecture representing a concrete example for a dynamic test case generation and execution scenario.

2 Related Work – IoT-A ARM

The ARM involves identification, use and specification of standardized components, understanding the underlying business models and deriving protocols and interfaces. The ARM is envisaged as a combination of a *Reference Model* (RM) and a RA. The RM aims at establishing a common understanding of the IoT domain with a set of models identifying the main concepts of the IoT, their interactions and constraints. It also serves as a basis of the RA. The RM, detailed in [1], includes, in particular, a *Domain Model* (DM) as a top-level description of IoT concepts and an *Information Model* (IM) explaining how IoT information is going to be modeled. The DM identifies the concepts of entities, resources and services as important actors in IoT scenarios. A short description of the concepts can be found in Table 1.

The RA aims at guidelines, best practices, views and perspectives that can be used for building fully interoperable concrete IoT architectures and systems. Once the RA is defined, it can be used by multiple organizations to implement compliant IoT architectures in specific application domains. The RA, through a functional view, provides the key *Functional Groups* (FG) potentially needed by any concrete IoT architecture, as shown in Fig. 2 (square rectangles in light blue).

The Application FG describes the functionalities provided by applications that are built on top of an implementation of the IoT-A architecture. The IoT Business Process Management FG provides an environment for the modeling of IoT-aware business processes which can be executed in the process execution component. Orchestration and access of IoT Services to external entities and services is organized by the Service Organization FG. It provides a set of functionalities and APIs to expose and compose IoT Services so that they become available to external entities and can be composed by them. The *Virtual Entity* (VE) FG contains functionality to associate VEs to relevant services as well as a means to search for such services. When queried about a service of a particular VE, this FG will return addresses of the services related to this

[1] IoT.est: IoT Environment for Service Creation and Testing (`http://ict-iotest.eu/iotest/`).

[2] IoT-A: Internet of Things Architecture (`http://www.iot-a.eu/public`).

particular VE (through the VE Resolution component). The corresponding associations between a VE and services that can be relevant to the VE are derived and maintained by the VE and IoT service monitoring component. The IoT Service FG provides the functionalities required by services for processing information and for notifying application software and services about events related to resources and corresponding VEs. The IoT service component defines service descriptions. The IoT service resolution component stores and retrieves information about a service and can be used to insert and update service descriptions, retrieve stored descriptions and provide the address of a given service. The Device FG provides the set of methods and primitives for device connectivity and communication.

The Management FG is tasked with efficient management of computational resources, with the *Quality of Service* (QoS) Manager ensuring the consistency of the expressed QoS requirements and the Device Manager setting a default device configuration and firmware upgrades. The Security FG ensures that aspects of security functions are consistently applied by the different FGs.

3 Requirements

Most of the requirements (function and non-functional) identified by IoT.est come from scenario analysis and are mapped to various categories (see below). Specific requirements related to the service life-cycle have also been also identified. After introducing the IoT.est requirements, we will show how the IoT-A ARM takes them into account.

3.1 IoT Requirements

We start talking about IoT when objects of the *Real World* (RW objects) have the capability to interact and cooperate with each other and/or when users can interact with those RW objects. Important aspects of IoT are therefore sensor and actuator technologies that allow to interact with RW objects or the environment and which alternatively can provide RW objects with perception capabilities. Many scenario-driven requirements in IoT.est cover this intrinsic characteristic of IoT. Other requirements specifically deal with the heterogeneous nature of IoT and have lot of impacts upon the architecture of IoT.est. Finally when facing a potentially extremely high number of objects and therefore sensors, actuators and tags, an IoT architecture must provide very powerful and efficient tools for discovering the entities that can be used to create an IoT service.

3.2 Requirements for Knowledge-Based Service Lifecycle Management

To support automated, dynamic IoT service creation and provisioning, the architecture should meet both design-time and run-time requirements. The design-time is understood as service creation time that splits into modeling and development phases.

To ease service creation a semantics-based knowledge-driven approach is suggested. The semantic descriptions represent the knowledge about reusable service components and their composition to complex services in a machine interpretable way to allow automated inference while being explicit, i.e. linked to a human-readable knowledge base, to ease maintenance. The descriptions must allow representation of *Service Level Agreements* (SLAs) to support business contract enforcement, service states such as *Quality of Service* (QoS), *Quality of Information* (QoI) and definition of business and technical roles related to authorization and service lifecycle management. Definition of usage conditions to allow trigger mechanisms in case of failure conditions is also a required characteristic of service descriptions.

To overcome the semantic gap between the sensor data and the IoT enabled business processes, the system must be able to collect low level sensed data and process it to high level information. This requires methods for knowledge based composition of the processing chain including reusable components for data aggregation, sensor fusion, pattern recognition, event detection and reasoning. The SCE must be run-time environment agnostic. It should allow specification of patterns and run-time constraints that match user goals. This requires methods for component capability representation and component discovery.

Run-time aspects cover service provisioning that can be subdivided into service deployment and service execution. Each phase of the service life cycle has to be supported by the corresponding test, monitoring and self-adaptation mechanisms. If one of the components in the system fails, the system should identify and provide alternative components. This imposes high demands on the time constraints of any knowledge-based approach. This requires methods for pre-computed fall back solutions and look ahead approaches.

3.3 Requirements for Test-Driven Service Creation

From a test perspective, the knowledge based life cycle management approach offers the ability to (semi-) automate the derivation of tests from the semantic service description. The semantic service description needs to contain detailed information about the service interface description; the IoT resource interface description and information about the desired SLA and the required meta-data for service provisioning (e.g., resources of desired service run-time environment). *Input Output Precondition Effects* (IOPE) are commonly known as the information required for describing the functional behavior of the service interfaces. For automated test derivation it is required that the IOPE information are stored in the semantic description. Contrary to classical web service testing approaches, the heterogeneity of possible involved IoT resources require a detailed description how to communicate with them.

The involvement of IoT resources changes how testing can be applied to these semantically described services: risk of costs and real world affects results in the requirement to execute the *System Under Test* (SUT) in a controllable environment (sandbox) to assure correct behavior before deploying the SUT in the real world.

3.4 Comparison of Reviewed Architectures against Identified Requirements

In this section we give an overview of the most important requirements from IoT.est per category (architecture, semantic, testing, service composition) and show how they relate to IoT-A requirements.

- ***Architecture, Interoperability, Scalability, Portability:*** Most of the requirements covered in this category come from the scenario analysis and are covered by IoT-A. Worth noting are heterogeneity (device, technologies, access), semantic support (see below), sensing and actuation support, support for monitoring and management (covered by the Management FG in IoT-A), security (generally compliant with IoT-A Security, Trust and Privacy model). In IoT-A, interoperability, Scalability and Portability (and also Security) are considered as qualities of the system and are described as *Perspectives* following the terminology of Rozanski *et al.* [2]. The ARM provides a large set of Design Choices which give the architectures a large number of options in order to fulfill any targeted quality of the IoT system.
- ***Semantics:*** In IoT-A, semantics is handled mainly in the IM of the RM, where it is explicitly shown that entities like Devices, Resources and Services are associated with Service Descriptions. How to handle the Semantic aspect of those descriptions is left to the designer, but IoT-A strongly recommends the use of RDF or OWL. Semantic registration and look-up are equally considered in IoT-A and are part of the Functional Decomposition. In IoT-A VE (corresponding to *Entity of Interest* (EoI) in IoT.est) may be described semantically as well. IoT-A requirements do not consider testing activities as part of the model so few IoT.est requirements on this specific topic are not explicitly covered;
- ***Service Composition:*** The set of requirements in this category are either relating to the SCE design and non-functional requirement (which is beyond the level of detail of the ARM), or describing some aspects of Service Composition that are implicitly covered by the Semantic or finally directly linked to the testing requirements which are not covered by the ARM;
- ***Testing:*** Majority of the requirements in this category are very much testing-specific and are –as expected- not considered in IoT-A. Some requirements go very precisely in the content of Service Description specifying e.g. bandwidth constraints, pre- and post-test actions, while IoT-A is only proposing a general scheme for modeling such descriptions as part of the IM;
- ***Other:*** Some requirements relating to role–driven or –based activities access control and confidentiality or specific security technologies are compatible with the Security and Privacy model of IoT-A. Finally, technologies pertaining to SOA-related requirements are also compatible and handled by the ARM (e.g. compatibility with WS-*, RESTful technologies).

4 Architecture

4.1 Architecture for IoT Service Creation and Provision

The architecture has to support the heterogeneity of the current IoT infrastructures and has to bridge the gap between low-level IoT services and high-level business

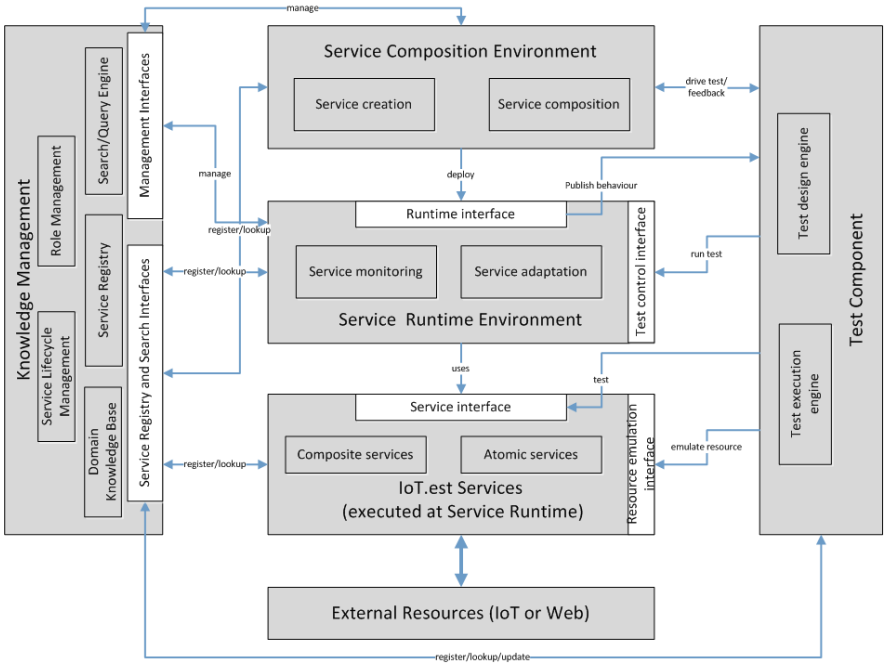


Fig. 1. Architecture for IoT Service Creation and Provision

services. Fig. 1 shows the IoT.est architecture for IoT service creation and provisioning. The objectives of the six major architectural components are summarized as follows:

The *Knowledge Management* (KM) component is responsible for registration, storage, search and query of the (IoT-based) service descriptions as well as some management tasks. The semantic service descriptions are stored in distributed service registries. The *Service Composition Environment* (SCE) allows end users to compose IoT based services in a semi-automated way based on business goals. A service design GUI facilitates the service creation with operations to build workflows and to edit data flows. It supports the entire service composition life-cycle, from supporting the process specification through elaboration of process and template expansions to the discovery and binding of service endpoints as activities within the processes. The *Service Runtime Environment* (SRE) enables provisioning of IoT enabled business processes. It is related to the deployment and execution phases of the service life-cycle. The SRE monitors context and network parameter (e.g. QoS) and initiates automated service adaptation in order to fulfill SLAs. The *Test Component* (TC) manages the derivation and execution of tests of the semantically described services. The test derivation is triggered by the SCE. It fetches the service description from the Registry and search/query engine where it also stores information about its test results. It handles the testing of the service in a controllable instance of the SRE, a so called Sandbox Instance, and emulates the external Resources. The IoT.est Services

component represents the collection of the IoT services and reusable service components. Since the target is to allow IoT specific services to be described, annotated and bound in a uniform manner, the term service is generic and not linked to any fixed options regarding the protocol, input/output format or any other specific SLA details. The External Resources are those not designed and developed within the proposed architecture. The resources can be services which can be discovered and used with IoT based services for service composition.

4.2 Reverse Mapping with Respect to IoT-A Domain Model

The list of concepts developed in IoT.est is at a lower level of detail than IoT-A. Obviously, concepts like resources, sensors, actuators are present but they are considered as 'External Resources', as the project it-self tends to focus on the service layer. Even if not developed here, the UML Domain Model from the IoT.est perspective would be compatible with the IoT-A DM. It would however feature a smaller number of main "concepts" but more sub-class entities (as many IoT-A concepts fall under the 'external resource' umbrella).

The list of main concepts used for the IoT-A / IoT.est DM can be found in Table 1:

Table 1. Mapping between IoT-A and IoT.est concepts

IoT-A Concept	Definition	IoT.est counterpart
User	The user of a service	User / Application
Physical Entity (PE)	Any 'things' in the real world including e.g. locations	Things
Virtual Entity (VE)	Representation of PEs in the digital world	Entity of Interest
Augmented Entity (AE)	Composition of VE and PEs. They are the 'Things' in IoT	n/a
Devices	Are used to mediate between PEs and VEs. Sensors, actuators and tags are typical devices	External resource
Resources	Resources are software components that encapsulate devices (resources can be quite Hardware dependent)	External resource
Services	Services offer standardized interfaces to the Resources	IoT service

IoT.est introduces the notion of Network Emulation and Resource Emulation that are used for encapsulating Resources in a black box for testing purpose. Those concepts are not present in the IoT-A DM. In the same way, the DM does not consider 'interface' as a concept at the current stage of its development.

4.3 Extension against IoT-A Reference Architecture

The proposed IoT.est architecture extends the IoT-A RA to include testing, run-time monitoring and adaptation as well as knowledge based service life-cycle management. Fig. 2 shows the major extensions and mapping to the IoT-A RA:

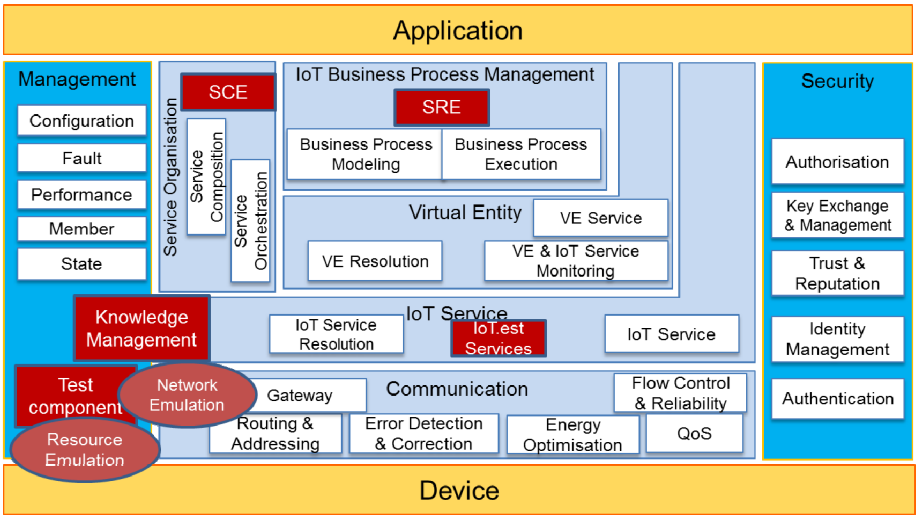


Fig. 2. Mapping of IoT.est functional blocks to IoT-A RA

The proposed framework defines a new functional block, the *Test Component* (TC), which can be situated as an additional component within the IoT-A Management FG. Based on a SOA approach, each IoT resource is wrapped by an atomic IoT service, which may be composed to more complex IoT services. TC also includes IoT Resource Emulation (to insert data transmissions from emulated resources into the SUT) and Network Emulation (to change the network connection behavior), which are defined at design time. Moreover, before deployment in the live system, the service is tested in a sandbox environment.

The KM component maps to the IoT Service FG with its service registration, search and query capabilities. Additionally, it maps to the Management FG with its lifecycle and role management functionalities. The IoT-A IoT Service FG also captures the IoT.est Services component, encompassing description and annotation of both atomic and composite services. The IoT.est SCE Component offers the same functionalities as the Service Organisation FG. The SRE functionalities map to the IoT Business Process Management FG, including built-in monitoring and adaptation capabilities to cope with the dynamics of IoT environments. A service starts monitoring the *Key Performance Indicators* (KPIs) once contracted. If the service SLA is violated, it may be necessary to trigger some adaptation actions to correct the deviations. Service Adaptation provides the necessary functionality (such as

reconfiguration, re-assignments and/or temporary work-around) to resolve the problem and get the service back to a normal operational state as efficiently as possible.

5 IoT.est Architecture Building Blocks

As a focused contribution to the identified requirements for a test-driven SCE, in this section, we concentrate on the Knowledge Management and Test Components of the proposed architecture.

The Knowledge Management block is based upon the IoT.est description ontology which is designed as the semantic description framework for the proposed architecture. It provides a light-weight modular framework for describing different aspects such as resources, test and services and is detailed in [3]. The services and resources are linked not only to the concepts in the domain knowledge base but also to other existing sources on the Semantic Web. The Service Description Repository is implemented as a distributed store which hides the complexity of distributed query processing from applications. It exposes management interfaces for service description registration, update, lookup, removal and discovery. The Search/Query Engine consists of two major sub-components: service search and service recommendation. The search procedure is primarily based on a logic based approach (e.g., using the SPARQL language [4]) and enables users to generate expressive and effective logical queries in combination with the distributed repository mechanism. To find the most appropriate service instances in the context of the applications, e.g., service composition, an effective service recommendation function is implemented based on factors such as the users' needs and tasks, application context, and possibly the trustworthiness of the underlying IoT resources. Lifecycle Management provides means to the other modules to update all the information related to the service lifecycle, such as status updates For example, a service that has passed the validation tests consequently can be certified; the change of the lifecycle status of the service includes other information that proves that the service has passed all the tests. Role Management provides the typical operations that allow to create, update, delete or modify roles and to associate and describe the functionality that the user role is allowed to access. For example, an integration tester will be able to certify tests; but not a software developer.

Test Component: Due to the real world interaction and the lack of control of components involved in atomic and composite services, tests cannot be executed in a production environment. Therefore, the SUT will be placed in a so called sandbox, which emulates the target environment as realistically as possible – not only functionally, but also from a real world, e.g., network and resource oriented, point of view. To overcome current time and resource intense approaches we propose a code insertion methodology, which can be derived from the semantic description of the IoT based service [5]. The Test Execution Engine (TEE) controls the environment and executes the test cases. The test creation process is triggered by the SCE and the resulting test cases can be selected to be executed or modified based on expert knowledge. A detailed description of the testing approach can be found in [6]. The Test Design Engine (TDE) creates test cases for new and changed services and prepares their execution.

The test cases are described with the standardized Test Control Notation Version 3 (TTCN-3) [7] language. The test cases are enriched with test data generated based on the IOPE conditions of the semantic service description. A Test Case Compiler produces executable code from the test cases. The TEE is the central component to coordinate the test flow and takes care of the test execution and executes the SUT under controlled and emulated conditions of the target environment. The sandbox ensures that the SUT can be executed in a test environment and can be manipulated during the test execution. In addition, the separation between the TEE and the sandbox offers the ability to execute the tests in a distributed manner. The SUT interfaces are connected with the TEE and a network emulation interface and this enables that each message from or to the SUT can be manipulated in terms of delay and packet loss to evaluate the robustness. Run-time behavior changes are made by the execution runtime emulation that ensures the identification of potential SLA violations. The strict isolation of the SUT within the sandbox is realized by the resource emulation interface, which encapsulates interfaces to external Web Services or IoT resources.

6 Case Study

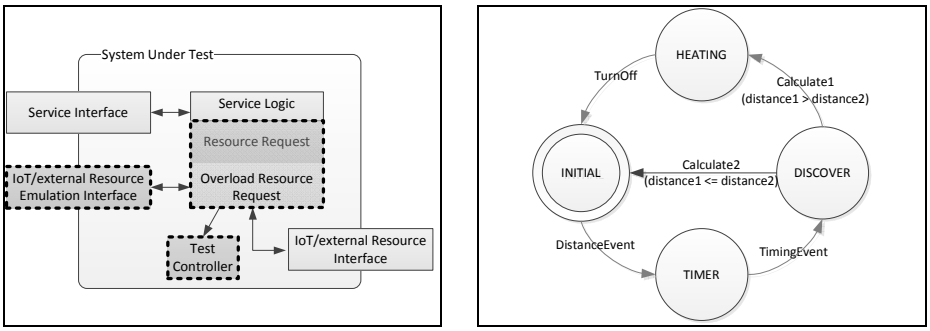
The concept of (semi-) automated test derivation is based on the machine interpretable semantic service description. The procedure of functional testing is explained using a simple example service. The goal of the service is to switch on the heating when the user approaches home. A *DistanceEvent* starts the service and the current distance is stored. On a *TimerEvent* a second distance is measured and the heating is switched on if the second distance is smaller than the first distance. Heating switch off can be realized by the user request *TurnOff* (cf. Figure 3(b)). In order to accomplish this, the service has to communicate with an external service to identify the user distance and communicate with the heater to switch it on.

For testing a behavior model of the service and an emulation of its interfaces to external services (i.e. distance measure) and IoT resources (i.e. heating) is required. The automated modeling of the service behavior and its interfaces is described as follows: *Derivation of behavior model:* The behavior model of the service is based on an extended *Finite State Machine* (FSM) that has input functions as transitions and an attached complex data model. Automated derivation of the behavior model exploits the combination of various knowledge, including business process description, semantic service description, and service grounding (e.g., full interface description and binding information). The objective of the model is to describe the service behavior for the test purpose in an abstract way. This paper suggests the use of rules (employing Rule Interchange Format Production Rule Dialect (RIF) [8]) to ease the formulation of the service transitions, i.e. preconditions and effects, by the tester. The steps to derive the service model are as follows: 1) state variables and their range of validity are identified (based on data types, or based on preconditions) 2) preliminary states are identified based on partitions of the state variable (e.g. precondition *distanceToHome <10 km*), 3) the input and output functions are determined based on reserved words in the rule descriptions and, 4) transition pre- and post-states are identified. The resulting

extended FSM is shown in Figure 3 (b) (without the data model). It shows, for example, that if the service is in the state *DISCOVERY,* the state is left by calling the input function *Calculate* either resulting in the post-state *HEATING* if the condition (distance1>distance2) is true, or in the post-state *INITIAL* otherwise.

Emulation of external services: To test the service in isolation, the connection to external (web-) services needs to be emulated. For messages from the SUT to external services the methods are encapsulated and if the SUT is in emulation mode, it sends the request via a RESTful interface to the emulation component. In this example, the resource emulation interface is capable of inserting the desired distances *(distance1 > distance2* and *distance1 < 10 km*) into the SUT.

Emulation of IoT resources: The access to IoT resources is emulated like the call to/from external services. Therefore the IoT interface is encapsulated also inside the SUT itself (cf. Figure 3(a) and [9] for more details). Two main steps are required to make use of this approach: 1) insert code based on the interface description and 2) create the behavior model in a similar manner as with the service interface. The inserted code enables a generic communication with the SUT and allows emulating the IoT resource behavior with data insertion. The inserted code consists of: 1) a singleton Test Controller class which stores the SUT mode (e.g. emulation mode), 2) a resource emulation interface, and 3) the encapsulation of methods that are communicating with IoT resources. The resource emulation interface handles all data insertions from emulated IoT resources and invokes the overloaded methods within the SUT.



(a) IoT Resource Emulation Interface (b) Simplified Finite State Machine

Fig. 3. IoT Resource Emulation Concept and Simplified Finite State Machine

With the creation of the resource emulation interface and the SUT model, the concrete test cases can be created. Based on the desired test coverage (here state-based for presentability) paths searching algorithms extract test cases from the extended FSM employing termination conditions to avoid endless loops. In this example, the resulting test cases involve the paths from the INITIAL state to the HEATING state. The created test cases are coded with the TTCN-3 language. The TTworkbench [10]

controls the test execution: The test execution starts with initialization of the service and the emulation components. Then the test case execution engine initiates the *DistanceEvent* by inserting a *distance1 < 10 km* into the SUT via the resource emulation interface. After the timer event has occurred, the *distance2 < distance1* is also inserted into the SUT. The reaction of the service ("turn heat on" message, intercepted by the overloaded methods inside the SUT) indicates the correct behavior of the service.

7 Conclusions

The concept of IoT services that are able to expose capabilities of their corresponding resources defines the paradigm of service-oriented computing in IoT. The mobile, unreliable, and capability-constrained nature of such resources make the IoT services different from most existing legacy services on the Web. In this paper, we have proposed a semantics-oriented test-driven architecture for a dynamic service creation environment for the IoT that addresses these issues in a coherent fashion. After deriving the generic and test-driven requirements placed on such an architecture, we mapped the envisioned building blocks to the IoT-A ARM in order to show compatibility with known IoT reference architectures. We also briefly present an implementation of the architecture components with an automated test generation case study.

References

1. Magerkurth, C. (ed.): Deliverable D1.4 – Converged architectural reference model for the IoT v2.0 (IoT-A Public Deliverable) (2012)
2. Rozanski, N., Woods, E.: Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives, 2nd edn. Addison-Wesley (2005)
3. Wang, W., De, S., Toenjes, R., Reetz, E., Moessner, K.: A Comprehensive Ontology for Knowledge Representation in the Internet of Things. In: 11th IEEE International Conference on Ubiquitous Computing and Communications (IUCC 2012), Liverpool, UK, pp. 1793–1798 (2012)
4. W3C, SPARQL Query Language for RDF, W3C Recommendation (2008)
5. Wei, W. (ed.): D2.2 - Report on Reference Architecture For IoT Service Creation and Provision (IoT.est Public Deliverable) (2012)
6. Reetz, E.S., Kümper, D., Lehmann, A., Tönjes, R.: Test Driven Life Cycle Management for Internet of Things based Services: A Semantic Approach. In: The Fourth International Conference on Advances in System Testing and Validation Lifecycle (VALID 2012), pp. 21–27 (2012)
7. ETSI. The Testing and Test Control Notation Version 3 (TTCN-3). European Standard (ES) 201 873 (2002/2003), `http://www.ttcn-3.org`

8. RIF Production Rule Dialect, 2nd edn. (W3C Recommendation), http://www.w3.org/TR/rif-prd/
9. Reetz, E.S., Kuemper, D., Moessner, K., Tönjes, R.: How to Test IoT Services before Deploying them into Real World. In: Proc. 19th European Wireless Conference (EW 2013), Guildford, UK (2013) (accepted for publication)
10. Testing Technologies - Products, http://www.testingtech.com/products/ttworkbench.php