FABIoT: A Flexible Agent-Based Simulation Model for *IoT* Environments

Marco Pérez-Hernández, Badraddin Alturki, Stephan Reiff-Marganiec
Department of Informatics, University of Leicester
Leicester, United Kingdom
{meph1, baba1, srm13}@leicester.ac.uk

***Abstract*—The Internet of Things aims to digitize everyday physical objects by connecting them to the internet. As a result, cyber-physical environments of multiple sizes emerge, imposing new requirements on applications and software systems in regards support to heterogeneity and volatility. A challenging stage in the engineering of these systems is the validation. Although, there have been significant efforts to offer shared real-world testbeds, the simulations platforms are required to make the validation process cost and time effective. Existing simulation approaches only offer partial coverage to the key *IoT* environment characteristics, focus on communication or are specific for particular use cases and domains. In this paper, we propose a novel agent-based model that enables the simulation of the *IoT* systems with the key characteristics of an *IoT* environment. This model is designed to be flexible and adaptable to different experiments. Our approach introduces events in *IoT* environments as stochastic processes, enabling the evaluation of *IoT* systems under different conditions that otherwise would be time consuming and costly. We present the results of our experiments for evaluation of our model. These show that our proposal is a practical solution for the validation of *IoT* software systems, complementary to the real-world tests.**

***Index Terms*—IoT Simulation; Agent Based Modelling; IoT services; Smart Objects; Internet of Things Modelling.**

I. Introduction

The Internet of Things (IoT) [1] has become one of the most active areas in computer science and beyond in both research and business contexts. Although the existing internet is already diverse, *IoT* environments are more heterogeneous, connecting simple *IoT* devices and more advanced Smart Objects (*SOs*) with several hardware architectures, resources, software platforms, communication protocols, message formats and data repositories among others. It is expected that by 2020, there will be around 50 billion or more devices connected to the internet [2], [3].

The engineering of software systems and applications for *IoT* environments reveal a number of open challenges in the area. Particularly, the validation of such systems is complex because of three key differential characteristics of the *IoT* environments: the heterogeneity, volatility and the size variety. In order to validate software solutions, researchers use one or a combination of the techniques that include real-world and simulations. Standard simulation techniques cover partial characteristics of these systems at individual or network level e.g. [4]. Other *IoT* test-beds such as SmartSantander [5] provide a real infrastructure for operation and management of the cyber-physical devices with a defined architecture and approach for software development model, offering limited interaction over the base software of the involved devices. Multiple methods for validation of *IoT* applications have been used, highlighting particular aspects of *IoT* environments. Rarely, these approaches enable the incorporation of heterogeneity, instability and large quantities of *IoT* devices in order to validate scalability and adaptation of the *middleware* and specific *SO* software, at individual and collective level.

The definition of a simulation model that considers heterogeneity, volatility and medium/large scenarios are still to be proposed. Ideally, this model must enable the repetition of experiments and support the definition of decentralized architectures while enabling experimentation with the different combination of services offered by every *IoT* device.

In this paper, we propose a novel agent-based model that enables the simulation of the operation of an *IoT* system, focusing primarily on the collective behaviour and the characteristics of the *SOs* in ecosystems with large numbers of *SOs*. This model enables:

1) repeatable simulation of multiple *SOs* that communicate and cooperate as part of a system,
2) easy definition of multiple metrics to be monitored, and
3) definition of several types of random events that are incorporated to the *IoT* simulated environments.

The remaining is organized as follows, see section number in parenthesis: we continue with a background on validation and *IoT* systems (II), then we describe the *FABIoT* model (III) and its architecture (IV). Next, we show the process of using *FABIoT* (V) and present the experimental setup (VI), results (VII) and discussion (VIII). Finally, we describe related work (IX) and draw conclusions (X).

II. Background

A. IoT *Validation*

Validation in real-world testbeds is always preferred as they enable to replicate the cyber-physical conditions that are present in production environments. The major drawback is the cost. Despite the low cost of hardware platforms for *IoT* devices, not all researchers have access to a real testbed with a sufficient number of *IoT* devices. Besides, the configuration, management and running of the platforms supporting the test environments, is also time consuming. Therefore, usually, the

real-world settings are constrained and scenarios lack of some or all of the key differential characteristics of *IoT*, namely: the heterogeneity, volatility and medium/large size.

In the last years, a number of *IoT* experimental research facilities with support to medium/large number of devices have appeared, offering an infrastructure for evaluating solutions atop of the offered services. These platforms offer services that reduce the effort required to evaluate a particular solution, however, the price is that application developers must conform to a particular development and operation model. e.g. *IoT* devices are merely data feeders and cloud infrastructure concentrates data storage and processing. In case of evaluating decentralized architectures that require edge processing the usefulness of these platforms is reduced. This is a problem since the *IoT* research agenda includes the development of decentralized solutions able to run in a combination of fog, edge and cloud contexts and environments for evaluating these solutions are required.

Besides, simulations enable validation under multiple conditions, defining a model that offers a partial representation of the real-world. From the literature reviewed (See section IX), the approaches for simulation and the models used, focus on a reduced number of characteristics of the *IoT* environments. One of the most common uses of simulations is to validate systems' scalability to medium/large number of *IoT* devices, however other characteristics of heterogeneity of *IoT* devices and volatility e.g. *IoT* device mobility and network topology changes, among others; are barely considered.

B. Smart Objects, Agents and Services

One of the approaches for building the IoT vision is based on the concept of Smart Object (*SO*). In this approach, the SOs are the individual augmented things that together combine as a system to make IoT scenarios possible. Although the concept comes from the first decade of the century and the basis for the Smart Object-based IoT (SOb-IoT) might be from the work by Kortuem, Kawsar et al. [6] back in 2010, nowadays the SOb-IoT and the *SO* as a concept are still under construction. The agent paradigm provides powerful abstractions with direct mapping to the *SO* characteristics. An agent is defined by Wooldridge as "a computer system that is situated in some environment, and that is capable of autonomous action in order to meet its delegated objectives" [7]. It is clear from this definition that the agent involves a software system that works autonomously towards some objectives. In order to meet these objectives, agents sense the environment, react to it, take the initiative to carry out actions, interact and work with others.

Service and Agent computing paradigms have proven to be powerful and useful software building blocks in multiple contexts. For IoT, this is not an exception. On the one hand, Web services are particularly suitable for web application development as they provide inherent interoperability and reusability. On the other hand, agents and Multi-agent Systems (MAS) are instrumental in the development of autonomous and cooperating systems that are able to reason and proactively take actions. Inter-operability, autonomy and cooperation are fundamental to achieving the SOb-*IoT* vision. However, it is a challenge to ensure these paradigms are used together effectively, enhancing each other and taking advantage the existing common grounds.

III. *FABIoT* Model Overview

FABIoT is an *Agent-based model* (*ABM*) that mimics the operation of different scale *IoT* systems over the time. *FABIoT* offers the toolkit for the definition of *IoT* environments and event-driven scenarios. The aim of *FABIoT* is to enable evaluation of distributed software systems that are intended to be installed in *IoT* devices. These systems include, for example, a collection of algorithms, services, *middleware* or a protocol.

Thus, *FABIoT* enables to create *IoT* spaces with multiple heterogeneous, distributed and connected *IoT* devices. These *IoT* devices are linked to a software library containing the implemented system to evaluate. Having the *IoT* space and the system to evaluate, *FABIoT* enables to simulate different and many scenarios incorporating not only stable but also volatile situations that, if done in real-world settings, would be a time consuming and costly effort.

In *ABM*, there are three fundamental elements: agents, environment and relationships [8]. The agents are embedded in the environment where they work autonomously during a time frame which is measured in ticks. In *FABIoT*, agents represent *IoT* devices, *Smart Objects*, *cyber physical* infrastructure and humans interacting with them. *SOs* are *IoT* devices with autonomous behaviour. In our case, the environment represents a physical setting where agents are placed, it could be a residential building, a factory, a school or a neighbourhood. Evaluations using *FABIoT* consider several characteristics of the *IoT* devices that can be defined as parameters or variables in every particular evaluation scenario.

The key characteristics that make *FABIoT* advantageous for evaluation of *IoT* solutions are indicated below. Note that some of these are inherited from the standard *Agent-based* modelling.

- **Flexibility:** The model is not individually designed to test one solution but to mimic common characteristics of the *IoT* scenarios.
- **Scalability:** It allows for the representation of small, medium and large scale scenarios.
- **Heterogeneity-friendly:** It enables to setup scenarios where the *IoT* devices can have different hardware platforms, capabilities and hosted services.
- **Volatility-friendly:** It enables simulate scenarios where the *IoT* devices are mobile, consume (and use-up) resources —e.g. battery, storage— or can change the services offered.
- **Event and Data-driven:** The model enables the definition of a series of events that can be triggered by a simulation cycle.
- **Micro and macro levels:** The model provides temporal data about the resources consumed and the task executed by every individual *IoT* device as well as the results of the interactions and behaviour of the whole system.

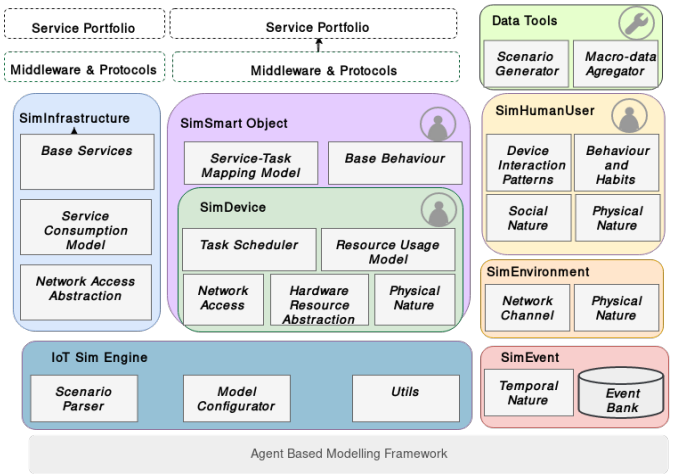

Figure. 1: *FABIoT* Model Architecture

- **Abstraction levels:** Software solutions can be evaluated at different abstraction levels. e.g. software and network infrastructure as well as applications.

IV. *FABIoT* Model Architecture

In this section, we describe the key elements of the model architecture as presented in Figure 1. The main type of agent in *FABIoT* is the *SimDevice* that represents an *IoT* device. The Algorithm 1 shows a high level view of the main operation of a *SimDevice*. This agent has a hardware and software platform and is able to sense/actuate from/on the environment. The *SimDevice* carries out tasks which are executed using its available hardware resources. The *Hardware Resource Abstraction* (HRA) component enables the definition of a different mix of resources including CPU, RAM, storage, battery, sensors, actuators, embedded interfaces such as displays or keyboards, among others. The *Resource Usage Model* (RUM) determines the way that tasks consume the resources available in the *SimDevice*. The *Task Scheduler* allocates tasks per time unit based on the hardware resources available for the device. It enables parametrization of the instructions the *SimDevice* runs per tick. Although the scheduling algorithm is a simplification of the real-world, it is designed to keep consistency considering the different hardware configurations. The speed of processing a task by a *SimDevice*, depends on the task size as well as on the device's configuration type. The more powerful the configuration type and the lower the size of the task, the quicker the *SimDevice* is able to process it. Battery and storage usage patterns are independent of the concrete tasks being processed. E.g. battery usage might depend on the hardware characteristics of the *SimDevice* such as the presence of a screen; likewise, storage might depend on the data to store, which might vary for different executions of the same task. Besides, the *Physical Nature component* allows for configuration of a variety of physical properties relevant to the scenario e.g. location, weight, size, etc.

On top of the *SimDevice*, another agent, the *SimSmartObject*, represents an *SO*. This is an entity that shares the hardware platform of the *SimDevice* but combines an autonomous behaviour and a software layer based on services. This software layer is generally the solution under evaluation. *FABIoT* allows for the definition of different mixes of services in a *Service Portfolio*. This portfolio determines which services are deployed in every *SO* involved in a scenario. Therefore, the heterogeneity is incorporated in *FABIoT* at hardware and software level.

Algorithm 1 *SimDevice*'s main function

1: **function** DOOPERATION(lts) ▷ lts:Last tick state
2: *cts.inMsg*← *receiveMsgs(simNetwork)*
3: *cts.tickBacklog* ← *lts.tickBacklog*
4: **for** *msg* ***in*** *cts.inMsg* **do**
5: *cts.tickBacklog* ← *processMsg(msg, protocolSpec)*
6: *cts.hardware*← *initializeHardware(lts.hardware)*
7: **while** *cts.hardware* $\neq \varnothing \wedge$ *size(cts.tickBacklog)* > 0 **do**
8: *nextTask* ← *pull(cts.Backlog)*
9: *execution* ← *schedule(nextTask, cts)* ▷ Also updates hardware resources according to resource model
10: **if** *execution* = *full* **then**
11: *executeTask(newTask, executionSpec)*
12: **else if** *execution* = *partial* **then**
13: *pendingTask* ← *executePartial(newTask, executionSpec)*
14: *cts.Backlog* ← *push(pendingTask, cts.Backlog)*
15: **else**
16: *cts.Backlog* ← *push(newTask, cts.Backlog)*
17: **return** cts ▷ cts: Current tick state

Another type of agent is the *SimHumanUser*, that represents the human user. This agent enables the configuration of social and physical properties of several users, as well as their behaviour within the environment and in relation to the *IoT* devices and *SOs*. These agents can trigger events or be affected by the *SimDevices* and *SimSmartObjects*. They activate functionalities under evaluation through the simulated interaction. Another component of *FABIoT* is the *SimInfrastructure* that represents any other computing resources —e.g. cloud or edge servers— that have powerful and sometimes unlimited resources with room to allocate and deal with more complex and resource-consuming tasks. Initially, the focus of *FABIoT* is in the behaviour of the *IoT* devices (micro level) and the whole system (macro level) rather than the infrastructure operation. Therefore, the infrastructure is abstracted in terms of services offered to the *SimDevices* and *SimSmartObjects*. The rules for usage of the available services are defined in the *Service Consumption Model*.

The *SimEnvironment* component of *FABIoT* defines a *Network channel* to which *SimDevices*, *SimSmartObjects* and *SimInfrastructure* have access to. The *SimDevices* use a standard request/response mechanism over a shared channel. The *SimDevices* can only send/receive messages from others connected to the same network. The message processing

function, at each end, consumes hardware resources and is also considered as a task for each *SimDevice*.

The *IoT Sim Engine* is the module in charge of articulating all the components of a scenario and manage its execution while collecting the relevant data for analysis. This engine provides a *Scenario Parser* that gathers the agents, environment and events for a particular scenario from files with a specific format e.g. CSV. The *Model Configurator* sets up every entity of the model and triggers initialization routines.

Two separate tools ease the generation of the input data required for the model to operate. The *Scenario Generator* produces pseudo random data sets containing entities, agents, relationships and events. Besides, the *Macro-Data Aggregator* offers routines for extraction of data from the time series for each run of the scenario.

V. Evaluation of *IoT* Software Systems using *FABIoT*

This section presents the process for evaluation of an *IoT* software system using *FABIoT* as depicted in Figure 2. This is a cycle inspired by the Montecarlo methods, which are useful for predicting variability of complex systems [9]. The rationale is that the behaviour of the different elements of an *IoT* system can be modelled as a stochastic process. The

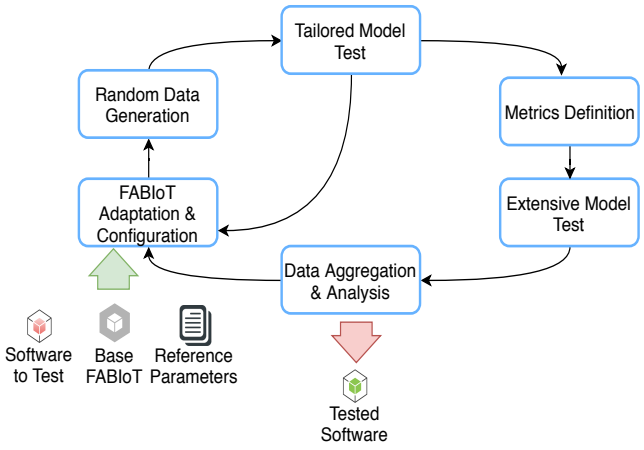

Figure. 2: *ABM* Process for validation of *IoT* software systems

randomness comes from the dynamics, heterogeneity and the flexibility of the *IoT* devices and the networks they built. Besides, there are a number of events that happen during the life cycle of the *IoT* system, making its response variable to the specific context conditions. For example, events can be an *IoT* device (*SimDevice*) joining, leaving the network —e.g. it runs out of battery— or sensing a stimulus from the environment (*SimEnvironment*). We define an observation period —i.e. the time range the *FABIoT* model runs— during which, several events can be configured to happen. *FABIoT* enables to gather data about the behaviour of the *IoT* devices, the *SOs* and a whole *IoT* system of them, working with the software system under evaluation. This is useful to evaluate its performance and identify potential issues facing unforeseen events.

The process in Figure 2 starts with the adaptation and configuration of the *FABIoT* base model. *FABIoT* works as a template offering various key elements of the *IoT* environment but, minimally, needs to be linked to the software to test. The result is a tailored model that is then parameterized. The parameters to configure are common to every scenario and cover the available hardware resources, the service-task mapping, the resource usage and the service consumption model. Although these parameters can be kept stable along different runs, this configuration gives the flexibility to introduce different behaviours in the *SimDevices* or *SimSmartObjects* as per requirement of the evaluation scenarios. As usual in *ABM* models the definition of parameters requires tuning against real-world subjects. In the case of hardware resources, references are based on available real configurations and can be further adjusted with benchmark from real-world small scale tests.

The process continues with the generation of the input data for the simulation. The random-generated data covers mainly the collection of events that will be triggered during the observation period and the physical, software and hardware characteristics of the *SimDevices* and *SimSmartObjects*. The data sets for the simulation cycles are generated using the *Scenario Generator tool*. The data for every event varies slightly, but includes minimally the event type, the tick it should happen and the *SimDevices* or *SimSmartObjects* involved. The synthetic data enables to test the software system without the need of a wide deployment, of the solution under evaluation, among the various different real *IoT* devices.

The next stage in the process is to test the tailored model, with a single small-scale scenario. If required, the model is re-configured and adjusted until it is ready for the extensive simulation cycles. This test provides examples of the output data, in the context of the scenarios and software under evaluation, that is used for the following step: defining the metrics relevant to study. Multiple metrics can be defined for comparing the system performance along different simulation cycles. Using *FABIoT*, time series data are obtained at both micro-level, i.e. the operation of particular *SimDevices* or *SimSmartObjects*, and at macro-level, the whole system behaviour. At individual level, for example, the number of resources used for a scenario provide insights about how the workload is distributed and which *IoT* devices are participating in a workflow. *FABIoT* also provides data about how *SimSmartObjects* manage situations when they, or their neighbours, consume the available resources. At collective level, network attributes such as latency and the overall usage of resources can also be measured. With the metrics, extensive model tests are performed, simulating the different conditions as per data generated and changing the relevant controlled parameters for the software under evaluation. Finally, the individual and collective data are aggregated and interpreted, if exit conditions for evaluation of the software are met the cycle ends, otherwise the model is adjusted and evaluated again following the same approach.

VI. Implementation & Experiments

The aim of the experiments was primarily to determine the feasibility of *FABIoT* for the evaluation of an *IoT* software system. As per the process described in Section V, we need a base *FABIoT* model and a software to evaluate. For the former, we implemented a prototype of *FABIoT* using the Repast[1] agent-based simulation platform. This platform provides general ABM abstractions such as agent, environment, patches, observer and a tool set for developing ABM models. We implemented a release of *FABIoT* including the *SimDevices* and *SimSmartObjects* agent types, the *IoT Sim Engine*, *SimEnvironment*, *SimInfrastructure*, *SimEvent* and the Data Tools (Section IV). For the *IoT* software system, we used a *middleware* solution that we developed in a previous work [10], [11]. This *IoT middleware* provides the set of abstractions for an *SO*-based software development and a p2p communication protocol. The main functionality of the *middleware* enables the *SOs* to:

- create overlay networks where they offer, query for and consume *IoT* services.
- put together related services offered by a particular *SO* in *roles* representing broader functionalities. The proportion of services per *role* indicates how many services a *role* is grouping.
- use services for realizing *activities* of *plans* that represent concrete workflows for achieving individual or cooperative *goals*.
- make decisions about the *SOs* to cooperate with, based on the *roles* they play and the current context.

For demonstrating the feasibility of *FABIoT* we focus on showing the performance of the *IoT middleware* using *FABIoT* with these two experiments:

- *EX1: Using* FABIoT *to simulate increasing workload*. It shows how a system of *SOs*, using the *middleware*, behaves when the workload is increased. We defined a one-to-one relationship between the *activity* of a *plan* and the service that realizes it. To show the effect of *role* grouping, brought by the *middleware*, we run the experiment for two proportions: 1 role grouping 1 service/*activity* (No *role* grouping) and 1 role grouping 5 services/*activities*. We generated data for 50 random configurations with a stable number of *SOs* (40), different hardware resources (See table I) and the events to simulate an increasing workload, in the range from 10 to 90 *plans*, where each *plan* had 7 *activities* to run within an observation period of 1000 ticks.
- EX2: Using *FABIoT* model to simulate departing *SOs* and evaluate system adaptation/resilience. It shows how the system of *SOs*, using the *middleware*, adapts when various *SOs* depart from the network. To see the effect of *service density* —i.e. how many *SOs* are offering a particular service— in the system response, we defined a case in which 10% of the *SOs* were offering the services (required to complete the scenario) and another case with 30% of the *SOs*. We generated data for 50 random configurations with a variable number of *SOs*. The *IoT* system is initially composed of 40 heterogeneous *SOs* and then the simulated events caused the departure of 20% up to 60% of these initial *SOs*. In the meantime, the system of *SOs* needs to complete a stable workload of 10 *plans*, within an observation period of 1000 ticks.

TABLE I: Main Model Parameters

Name	Description	Scope	Values	Units
Hardware				
Processing Power	Maximun amount (millions) of instructions the *SO* is able to process in a time unit (tick).	Model	1	Mipt
Config type	A combination of: No. of Cores, RAM and Storage, respectively.	Model	A (1, 0.5, 2) B (2, 1, 16) C (4, 2, 32) D (4, 4, 64)	Cores: Units RAM: Gb Storage: Gb
SO per type	Percentage of *SOs* per Config type	Experiment	[1-100]	%
Battery	Battery powered	*SO* type	[0, 1]	(Discrete)
Network				
TTL	Time-to-live for messages	Model	4	hops
PING Frequency	How often *SOs* PING others	Model	30	ticks

For the configuration of *FABIoT* we define the values for parameters presented in Table I. For each parameter, there are two possible scopes, those that are stable for the whole model and others that vary according to the experiment. For example, the four hardware configuration types are defined at model level, but how many *SOs* of each type can be defined per experiment. Some of these parameters come from state-of-the-art references —e.g. hardware configurations—, in other cases we came up with the relevant values after individual test and tuning. Finally, we executed the simulation cycles for each experiment using the ALICE[2] High Performance Computing Facility at the University of Leicester.

VII. Results

We defined two metrics to measure the performance of the system of *SOs*, using the *IoT middleware*: *Mean Query Time* (*MQT*) and *Plan Success Rate* (*PSR*).

MQT is calculated for the *plans* that the whole system completes and is an indicator of how quickly the *SOs* are able to locate other cooperating *SOs* available. *MQT* is a component of the total execution time of a *plan*, the rest of the time depends on the power and workload of each *SO*. The *MQT* is calculated:

$$MQT = \frac{\sum_{i=1}^{n} TQTP_i}{n}, \tag{1}$$

where $TQTP_i$ is the total query time for a *plan* i of n *plans* completed by the system of *SOs* during the period of analysis. *TQTP* is calculated as the sum of the query time of every *activity* of the *plan*.

Plan Success Rate is calculated as follows:

$$PSR = \frac{CP}{TP}, \tag{2}$$

where *CP* is the quantity of completed *plans* and *TP* is the quantity of triggered *plans* during one simulation cycle.

Figure 3 presents the *MQT* and *PSR* results of EX1. Each

[1]https://repast.github.io/index.html

[2]http://www2.le.ac.uk/offices/itservices/ithelp/services/hpc/alice/about

proportion of *role* to service/*activity* is identified with the different colours and shapes. The black shape shows the average in every case. The results show that the *MQT* decreases when the number of *plans* is increased. This is explained as the *middleware* enables the *SO* to cache other *SOs*, it has cooperated with, to avoid querying for a service every time the *SO* needs to execute an *activity* of a *plan*. It is clear that *MQT* values for the 1-to-5 proportion are slightly but consistently lower than the ones with no *role* grouping. This is explained, as the *middleware* enables *SOs* to reuse query results for services grouped by the same *role*, reducing the number of queries required and therefore the *MQT*. The *Plan Success Rate* results show that the increase in the workload barely affects the system of *SOs* for both proportions evaluated.
Figure 4 presents the *MQT* and *PSR* results of EX2. Each service density is identified with the different colours and shapes. The black shape shows the average in every case. The *MQT* results show clearly two different cases when the *service density* is 10% and when it is 30%. In the former case, the system of *SOs* takes more time to query for services when the number of *SOs* departing from the network increases, showing high sensibility to these changes. The latter case is less sensible, as the *MQT* remains almost stable when the density is 30%. The *Plan Success Rate* results show that the system of *SOs* copes well with departures of *SOs* when the *density* is 30%, keeping the *Plan Success Rate* higher than 40% and mainly around 70%, even with the 60% of *SOs* having departed. With the lower *density*, the *Plan Success Rate* is clearly lower than the previous case. These are expected results and show the *middleware* enable *SOs* to take advantage of the service offer to mitigate effects of volatility in the system.

VIII. Discussion

The experiments presented, show how we defined *IoT* scenarios using *FABIoT* and how we incorporated variability by increasing workload or removing *SOs* within these scenarios. *FABIoT* enabled us to evaluate the system, determining its tolerance thresholds and identifying improvements opportunities. Performing these tests in real-world setting would have been not only expensive but also time consuming. Being able to generate synthetic random data enable us to test the system even in unforeseen situations. with *FABIoT* we did not require to deploy the system under evaluation in a real-world setting to determine how it works. Since *FABIoT* represents key hardware and software characteristics of the *IoT* devices, it makes it easy to translate real-world configurations to the simulated environment. It also makes easy to generate representative situations for evaluation of the software system. We learned the importance of tuning *FABIoT* considering real-world reference parameters (Section V) for obtaining meaningful results. *FABIoT* is not intended to replace real-world test-beds but can be used as a practical complement, a previous step, in order to reduce effort, time and cost of evaluation of *IoT* systems.

IX. Related Work

There are several proposals for validation of *IoT* systems. We organize the works reviewed in two groups: real-world testbeds and simulation approaches. In this review, we explore a few examples of real-world testbeds and focus on simulation techniques, specially the existing agent-based approaches.

Real-word testbeds enable the use of a real infrastructure, usually built from a medium/large number of devices. Authors in [12] introduce FIT IoT-LAB which is a testbed with a large number and variety of nodes, distributed among 6 locations. FIT IoT-LAB enables evaluation and testing of solutions ranging from lower level protocols to advanced services and analytics in a large scale *IoT* environment. This testbed offers: remote access to gateways, sensors, and nodes via IP connection; monitoring of each devices power consumption and evaluation of real-time decision making in the context of 117 mobile robots. They incorporate heterogeneity at hardware level and provide tools for researchers to develop their own drivers, operating systems and applications. SmartSantander [5] is a testbed that also offers a large number of *IoT* devices distributed among diverse places, particularly, around the Santander city. It has a three-tier architecture including *IoT* nodes that sense noise and temperature, repeaters, gateways and sensors. They made possible to get the sensed data from the different nodes via different communication protocols. These platforms are robust and actively supported. As natural, these are shared by multiple researchers and they do not offer a clear approach for simulation of volatility conditions. In the real world, this is challenging as remote access and monitoring services can be compromised as well as other experiments running on the platform. In fact, FIT IoT-LAB team recognizes the importance of performing simulations before using their facilities.

Simulation approaches are wide, in most of the cases offering partial coverage of the key *IoT* characteristics identified in this paper. An agent-based simulation proposal using the *SO* concept is presented in [4]. They focus on inter-*SO* communication while enabling definition of subnetworks and recognition of network bottlenecks. The simulation is built on top of OMNET++ platform and present different scale scenarios. Since the focus is on communication, it is not clear the approach for simulating hardware/software heterogeneity, volatility nor the reuse of the model for other experiments. Authors of [13] present a simulator enabling visualization of an architecture for the Social-*IoT*. Their proposal includes a GUI where connections can be defined for nodes that are part of social networks made from device, brokers and human users agents. The solution is based on a cognitive middleware offering ontologies, agents and a publish/subscribe pattern communication. Another proposal is SimpleIoTSimulator [14] which is a commercial solution where users can create *IoT* environments including gateways and sensors. Besides HTTP, they support some of the popular *IoT* protocols such as MQTT and COAP. They enable the creation of scripts with ”error scenarios”. One of the limitations is the lack of support to

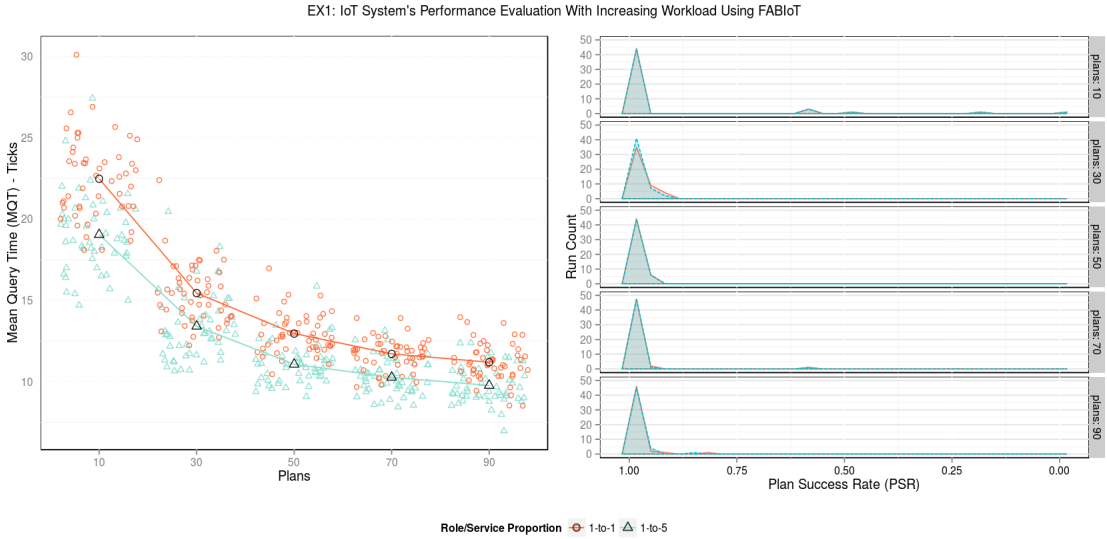

Figure. 3: EX1: Using *FABIoT* to simulate 50 different random network configurations (one per run) of an *IoT* system with 40 heterogeneous *SOs* facing workload increase (10 to 90 plans). It shows how: a)The *MQT* is inversely proportional to the workload. There is a slightly lower *MQT* for roles grouping five services/activities. b)The number of plans completed is not affected by the workload increase.

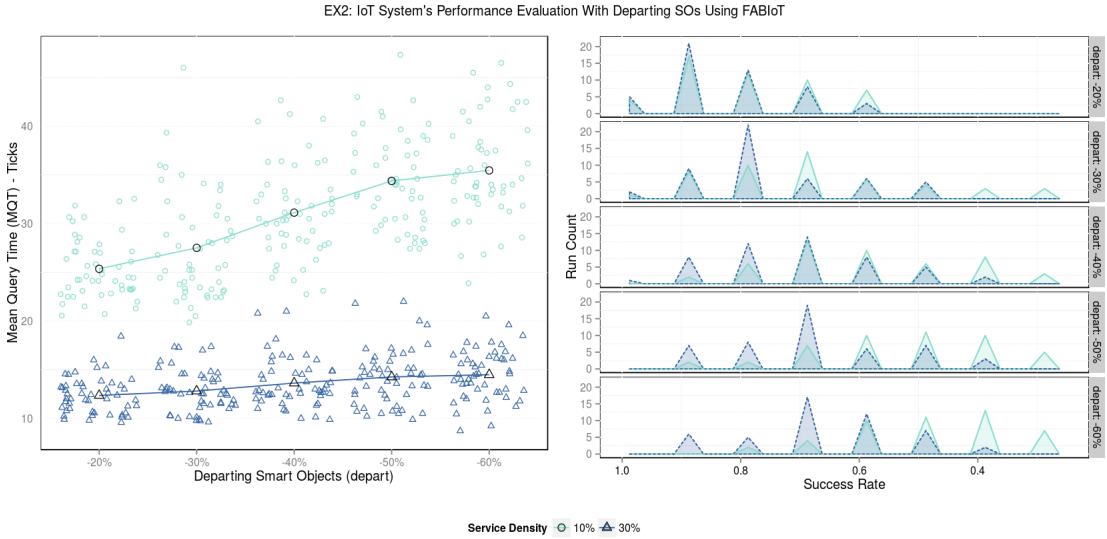

Figure. 4: EX2: Using *FABIoT* to simulate 50 different random network configurations (one per run) of an *IoT* system with 40 *SOs*, facing the departure of the 20% to 60% of the *SOs*. It shows how: a) The *MQT* remains stable for a high service/activity density whereas it raises for low density. b)The number of plans completed is reduced when *SOs* depart, particularly more sensible for low service/activity density.

SOs that are able to work autonomously and make decisions within the *IoT* environment. Authors of [15], present the concept and design aspects of Sensesim. Their focus is on simulating heterogeneous and autonomous networks. Sensors are autonomous agents composed of four classes: SensorLogic, SimEntity, SensorAPI and *middleware*. The SensorLogic class is to modify the sensor's state, the SimEntity for controlling the sensor as a simulation entity, the SensorAPI offers functions to interact with the sensor —e.g. start, stop, send messages— and the *middleware* that controls the execution of macro programs deployed on the sensor. A simulation platform and methodology, particularly suitable to test large *IoT* systems is proposed in [16]. The simulation supports distributed devices and enables simulation of *IoT* nodes with various network interfaces, mobility and consumption of energy.

Authors of [17] present the simulation of a smart grid system based on Multi-agent systems. They mainly focused on the device creation and energy consumption. Every device acts as an agent, enabling also the interaction with real devices and other existing agents. The paper in [18] shows an agent-based simulator built on top of the JADE Agent platform and employing Devices Profile for Web Services (DPWS). Every agent represents a SOA-ready device that uses the JADE infrastructure services to communicate to other agents. This

imposes the constraint on the simulated scenarios that require a Directory Facilitator (DF) for locating others instead of using more independent P2P approach. There is a layer that controls the scenarios called Superordinate logic that the user needs to perform the simulation. DPWS is also the base of a simulation toolkit for the design, development and testing of service-based *IoT* applications that is presented in [19]. This solution is based on the OASIS standard for DPWS that enables simulated devices to consume services following this standard. In this model, devices can be discovered and their operations invoked, while there is autonomous behaviour from their side. More recent proposals aim to simulate fog computing and IoT environments. One of these is iFogSim [20], which is able to simulate fog nodes and *IoT* environments via tuning the capabilities of devices including computational power, energy consumption, storage and communication.

Two solutions for Wireless Sensor Networks (WSN) simulations are presented in [21] and [22]. The first tool supports multiple-scale scenarios and an agent deals with events and devices, while there is an OpenStreetMap module that enables distribution of sensors, a WiSen Simulator organizes the simulation and a Solver controls the computations required for the simulation. The second proposal is an event-driven simulation with support for heterogeneity. It enables simulation of nodes with diverse energy resources, mobility models, applications and routing protocols as well as physical event simulation.

The proposals reviewed have made a good impact in research, however, we observe the lack of simulation tool addressing heterogeneity (at different levels), volatility, support to multiple scale scenarios, repeatability or induction of events, including those causing disruption to the *IoT* system. Validation in real-world testbeds without previous simulation is not cost-efficient and can be complex to implement.

X. Conclusion and Future Work

This paper presents a novel agent-based model that enables the simulation of the *IoT* system operation covering key characteristics of the *IoT* environments. This model allows for simulation of communication and cooperation of multiple *SOs* within an *IoT* system. We presented the architecture of our model and the simulation approach that includes the definition of randomly-triggered events that enable evaluation of *IoT* systems under different conditions. On top of this model, multiple metrics can be monitored and defined according to the experiments to perform. We evaluated our model with a series of experiments for the validation of an *IoT middleware* that is embedded in simulated Smart Objects.

On the basis of the promising findings presented in this paper, future work will involve simulation of *IoT* environments highlighting differences between *Fog* and *Cloud* computing contexts. Besides, we will assess the use of real data sets, when available. We would like to use data gathering patterns about device resource usage, e.g. energy consumption.

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, "Vision and challenges for realising the internet of things," *Cluster of European Research Projects on the Internet of Things, European Commision*, vol. 3, no. 3, pp. 34–36, 2010.

[3] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the internet of things: A survey," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 414–454, 2014.

[4] G. Fortino, W. Russo, and C. Savaglio, "Agent-oriented modeling and simulation of iot networks," in *Computer Science and Information Systems (FedCSIS), 2016 Federated Conference*. IEEE, 2016, pp. 1449–1452.

[5] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis *et al.*, "Smartsantander: Iot experimentation over a smart city testbed," *Computer Networks*, vol. 61, pp. 217–238, 2014.

[6] G. Kortuem, F. Kawsar, V. Sundramoorthy, and D. Fitton, "Smart objects as building blocks for the internet of things," *IEEE Internet Computing*, vol. 14, no. 1, pp. 44–51, 2010.

[7] M. Wooldridge, *Introduction to Multiagent Systems*, 2nd ed. Glasgow: Wiley & Sons, Ltd, 2009, vol. 30.

[8] C. M. MacAl and M. J. North, "Tutorial on agent-based modelling and simulation," *Journal of Simulation*, vol. 4, no. 3, pp. 151–162, 2010.

[9] B. Dodson, P. C. Hammett, and R. Klerx, *Probabilistic Design for Optimization and Robustness for Engineers*. Chichester, England; Hoboken, NJ: John Wiley & Sons, Ltd, 2014.

[10] M. E. P. Hernandez and S. Reiff-Marganiec, "Autonomous and Self Controlling Smart Objects for the Future Internet," *2015 3rd International Conference on Future Internet of Things and Cloud*, pp. 301–308, 2015.

[11] M. E. Perez Hernandez and S. Reiff-marganiec, "Towards a software framework for the autonomous internet of things," in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, 2016.

[12] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "Fit iot-lab: A large scale open experimental iot testbed," in *2015 IEEE 2nd (WF-IoT)*, Dec 2015, pp. 459–464.

[13] P. Kasnesis, L. Toumanidis, D. Kogias, C. Z. Patrikakis, and I. S. Venieris, "Assist: An agent-based siot simulator," in *2016 IEEE 3rd (WF-IoT)*, Dec 2016, pp. 353–358.

[14] simpleiotsimulator: the internet of things simulator. Accessed: 2018-02-19. [Online]. Available: http://www.smplsft.com/

[15] M. Dyk, A. Najgebauer, and D. Pierzchala, "Sensesim: An agent-based and discrete event simulator for wireless sensor networks and the internet of things," in *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec 2015, pp. 345–350.

[16] G. Brambilla, M. Picone, S. Cirani, M. Amoretti, and F. Zanichelli, "A simulation platform for large-scale internet of things scenarios in urban environments," in *Proceedings of the First International Conference on IoT in Urban Space*. ICST, 2014, pp. 50–55.

[17] S. Karnouskos and T. N. De Holanda, "Simulation of a smart grid city with software agents," in *Computer Modeling and Simulation, 2009. EMS'09. Third UKSim European Symposium on*. IEEE, 2009, pp. 424–429.

[18] S. Karnouskos and M. M. J. Tariq, "Using multi-agent systems to simulate dynamic infrastructures populated with large numbers of web service enabled devices," in *Autonomous Decentralized Systems, 2009. ISADS'09. International Symposium on*. IEEE, 2009, pp. 1–7.

[19] S. N. Han, G. M. Lee, N. Crespi, N. Van Luong, K. Heo, M. Brut, and P. Gatellier, "Dpwsim: A simulation toolkit for iot applications using devices profile for web services," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 544–547.

[20] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[21] K. Mehdi, M. Lounis, A. Bounceur, and T. Kechadi, "Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool," in *7th International ICST Conference on Simulation Tools and Techniques, Lisbon, Portugal, 17-19 March 2014*. ICST, 2014, pp. 126–131.

[22] G. Chelius, A. Fraboulet, and E. Fleury, "Wsnet: a modular event-driven wireless network simulator," 2006.