Integrating the EPCIS and Building Automation System into the Internet of Things: a Lightweight and Interoperable Approach

Nam Ky Giang[1]*, Janggwan Im[2], Daeyoung Kim[2], Markus Jung[3], and Wolfgang Kastner[3]
[1]*University of British Columbia, Vancouver, Canada*
kyng@ece.ubc.ca
[2]*Korea Advanced Institute of Science and Technology, Daejeon, Korea*
{limg00n, kimd}@kaist.ac.kr
[3]*Vienna University of Technology, Vienna, Austria*
{mjung, k}@auto.tuwien.ac.at

Abstract

The Electronic Product Code Information System (EPCIS) is a state of the art information system for tracking and monitoring the lifecycle of physical objects based on RFID technology. Low-cost passive RFID tags are replacing traditional barcodes for physical object identification and are heavily used within logistics and trade. They are an essential aspect of identifying physical objects, therefore an integration into the IoT is required. This paper presents a concept, which defines how the EPCIS can be integrated into the IoT extending it to a so called Smart Thing Information System (STIS). The integration follows a two-fold approach. On the one side, the information of the EPCIS is made available for IoT devices and services, by providing a lightweight query interface. On the other side, the information of non-RFID related devices, such as sensors and actuators of home and building automation systems (BAS) are made available to EPCIS. Thus, a seamless integration with building automation systems is possible. The extended system embraces interoperability by resting upon standards such as OBIX and CoAP. We present a case study using a globally testbed that demonstrates the feasibility of the proposed system extension. In addition, a performance analysis is carried out that investigates the scalability impacts and limits of the STIS new interfaces as well as the gateway for BAS.

Keywords: EPCIS, STIS, Building Automation System, IoT architecture

1 Introduction

The IoT is an emerging paradigm that extends the existing Internet infrastructure to the most constrained devices and the embedded world. Smart objects equipped with ICT-intelligence become citizens of a world wide Internet based communication infrastructure that interconnects appliances, sensors and actuators, mobile devices, wearables, identification tags, smart objects and cloud based information systems and services.

From the initial supply chain visibility application, the IoT has been evolving dramatically to embrace many other application domains such as building automation, pervasive health care or smart infrastructure. These evolutions require upgrades to the existing information infrastructures. Specifically, the traditional Electronic Product Code Information System (EPCIS) [1] which has been used extensively for objects management in manufacturing and supplying processes, needs a redesign to accommodate new IoT application domains with new requirements.

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications, volume: 6, number: 1, pp. 56-73
*Corresponding author: Room x509 ICICS Building, 2366 Main Mall, Vancouver, BC, Canada, V6T 1Z4, Tel: +1-778-322-2539, Web: http://www.resl.kaist.ac.kr/

Building Automation is a typical IoT application for these new requirements. A Building Automation System (BAS) usually contains many devices that possess fruitful data such as their working status or environment condition. These data are of much interest for maintenance process and thus, need to be indexed and managed efficiently in an information system that allows easy data retrieval and supports decision makings.

To address these requirements, IoT6 - a FP7 European research project [1] has proposed an extended version of EPCIS called Smart Things Information Service (STIS) [2]. STIS seamlessly integrates traditional EPC networks with a new breed of data-generating devices such as EnOcean [3], 6LoWPAN [4], and off-the-shelf BAS such as BACnet [5], KNX [6]. The basic idea behind STIS is an extensible capturing interface that can capture data from heterogeneous devices in addition to just RFID events. Leveraging the standardized EPC object extensions [1] STIS is able to store and managed the vast amount of data generated by things. In addition, STIS opens a lightweight query interface based on REST-ful web service [7] for easing the data retrieval from the system, thus facilitating the maintenance process and development of third-party applications. The newly added interfaces use CoAP protocol [8] for transportation and OBIX standard [9] for data format. The lightweight characteristics of the CoAP protocol and the robustness of the OBIX standard make them the best fit for the proposed system.

This paper extends the idea presented in [2] by discussing in detail the underlying techniques used in the two newly developed interfaces between STIS and BAS and giving a deeper performance analysis of the deployed system.

The paper is organized as follows. In the next section, we outline some related works in the field of IoT architecture design, EPCIS development and data exchange format. Section 3 describes some use cases in integrating the STIS with BAS. Section 4 introduces the proposed STIS system in details and Section 5 presents a case study that demonstrates the feasibility of the proposed concept. Further, a performance analysis of the involved components is conducted. Finally, Section 6 provides a conclusion of the presented integration of EPCIS and BAS.

2 Related Works

There are significant efforts in realizing an IoT architecture in which authors tried to design a platform for storing and retrieving thing's data [10] [11] [12]. In these works, physical devices are configured so that they periodically publish their contextual data to the platforms. The published data streams are then abstracted into channels to which client applications can subscribe to retrieve and process the data. However, there is a lack of a common data format standard so that it is virtually impossible to exploit physical thing's resources in large scale. In addition, these systems do not take into account the identity of physical devices. Instead, each thing is assigned a unique, random channel ID to identify itself when publishing data. Due to this approach, these systems do not benefit the product life-cycle management, which is very important for many applications. For example, the system will not be aware of which real product is generating which data stream, making product management and data analysis more difficult.

To overcome these limitations, this paper extends the EPC Information Service, which was standardized by GS1 Global [13] to handle things' data along with their identities. The proposed Smart Thing Information System (STIS) leverages the product life-cycle management of the existing EPC Network while provides functionality to capture the things' contextual data.

There are some other extensions of EPCIS that aim at providing more flexible and open interfaces for third party client applications. Leveraging Web Service architecture, authors in [14] proposed Web service interfaces to existing EPCIS to expose products information using the SOAP protocol. In another work [15], authors developed an extra server component that adapts the EPCIS repository to a RESTful

[1] http://iot6.eu

Web service, so that enables access to the EPCIS from conventional Web browsers. Beyond these works, our EPCIS is not only equipped with RESTful Web service interfaces but also extended to support the capture of things' contextual data. Moreover, our proposed extension of EPCIS is capable of communicating over the CoAP protocol, a new, lightweight application protocol for embedded devices. Thus, the EPCIS is now open to both client applications and embedded devices. This functionality opens a vast number of new application scenarios that can be developed. For example, physical things can subscribe to each others' data stream via STIS over CoAP and act upon each others' updates, thus enabling physical mash-up services.

Since the new version of EPCIS requires a standard in data communication between the system and the devices, a proper data format is indispensable. Within [16] and [17] the authors described how the heterogeneity found in nowadays home and BAS can be addressed through an integration layer based on OBIX and novel Web service protocol bindings to CoAP and JSON, which make a deployment of such communication stacks feasible for constrained micro-controller based sensors and actuators. The work presented in this paper extends this integration to RFID based information systems.

3 Use Cases for Integrating STIS with BAS

For combining the STIS with BAS, several use cases can be identified:

- **Building automation for smart logistics:** In an integrated solution, delicate goods (e.g., medical goods or food) with strong requirements on certain environmental conditions can be linked with the BAS of storage rooms in order to adjust temperature set points for the air condition. The product or more realistic the storage container can **control its environment** in a smart manner. Further, in case of violations of certain conditions, **alarms** could be **raised in the local BAS** which might include visual and acoustic signals and the notification of a system operator.
- **Auditing:** Sensor readings of the BAS can be logged by the STIS in order to proof the correct storage of goods throughout the complete supply chain.
- **STIS for building maintenance:** From a building management perspective, the integration of STIS opens the possibility to innovative use cases for building maintenance. Automation devices equipped with RFID tags would allow to automatically redirect a system operator to the smart things information service of the vendor allowing him to order spare parts in case of a device failure. The STIS can be further enriched with product data sheets that guide the installation process of devices.

For realizing these innovative use cases, a close integration between the extended EPC information system and building automation technologies is required to create a Smart Thing Information System for the future Internet of Things.

4 STIS Architecture

This section explains how the current EPCIS is extended with new features. Fig. 1 shows the architecture of the STIS that includes the original EPCIS. At the bottom, there is a a device layer where smart physical things reside. Raw data generated from this layer is filtered and collected into Filter and Collection (FnC) middleware, lies at the middle layer. At the top, there is a data repository that keeps all the filtered data and exposes them to client applications, where application logics run. The upgraded and extended interfaces are the Lightweight data querying interface and the new data capturing interface. These two interfaces are introduced as follows.

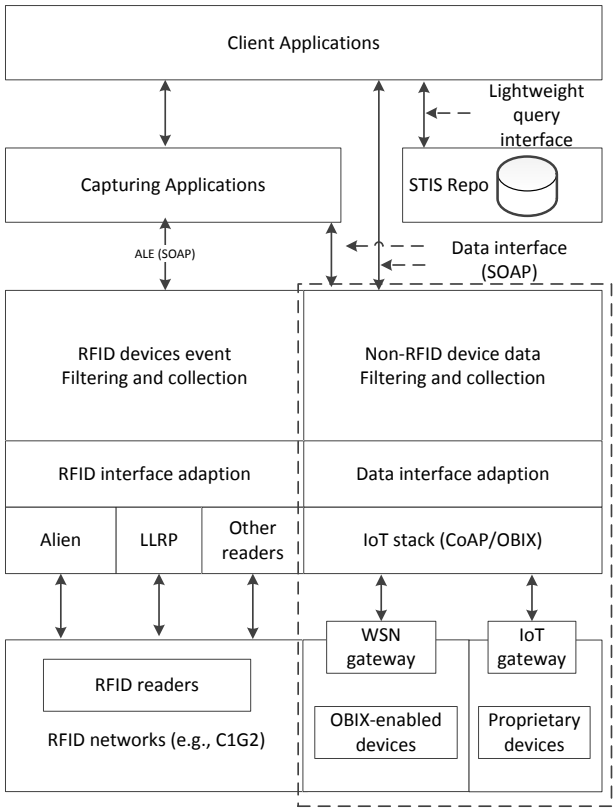

Figure 1: STIS components

4.1 Lightweight data querying interface

EPC Network has been designed for manufacturers and corporations who want a complete product management information system. Therefore, most of the interfaces between EPC Network components such as the FnC, the Repository, ONS or the Discovery module are based on heavy enterprise-level Web service interfaces such as SOAP/HTTP. With the evolution of the IoT, it is increasingly demanded that the product consumers be knowledgeable on the products they are consuming, e.g, how they are distributed or products' real origins. Furthermore, under the IoT context, objects are also interested in the captured data. For example, physical mashup services can be realized by having objects subscribe to each other's data streams and take corresponding actions. Therefore, we extend the EPCIS by designing a lightweight query interface that eases the data retrieval process of consumers or other physical objects (client applications).

The lightweight interface is based on the RESTful communication model that allows client applications to query for stored data through a simple resource URI. The interface's internal mechanism is illustrated in Fig. 2. As seen in the figure, only a portion of the database is REST-ified and made available to the lightweight interface. This is because client applications are usually interested in only the contextual data of the products. Meanwhile, enterprise applications are mostly interested in other EPC-related information, such as business locations, business steps and data point. Thus, the REST-ified portion of the EPCIS repository is grouped into context-oriented resource objects such as EPC event object, location object, EPC object. These resources are designed to give client applications a minimum set of

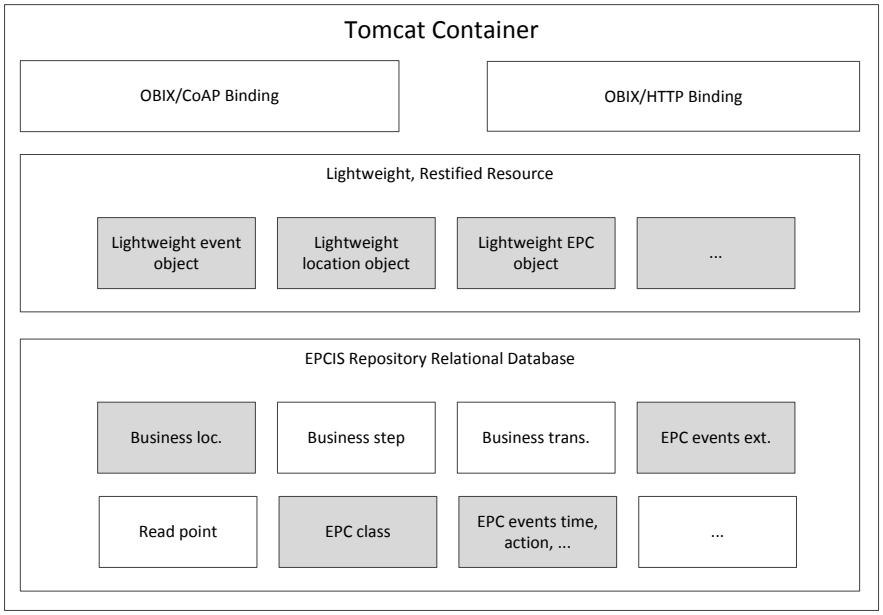

Figure 2: The STIS lightweight query interface

most necessary information. For example, lightweight EPC event object has only three properties: event timestamp, event action and the event data extension, which are useful for tracking the current contextual status of the object without knowing other information such as the data point or business location of the event. Besides, the lightweight location object gives information on only the data point location where the event occur.

To enable a variety of applications, STIS' lightweight query interface operates in both CoAP and HTTP protocol bindings. For instance, via the CoAP interface constrained objects can subscribe to other objects' data stream to act upon changes. Via the HTTP interface, product consumers can query the locations the products that have been shipped.

4.2 New data capturing interface

In STIS, in order to support capturing capability for sensing data a new data acquisition component is added to the *Filter and Collection (FnC)* layer of the traditional EPC network. This component is responsible for getting sensing data from things and convert them into legacy EPC events, which are eventually captured into the STIS repository.

The capturing component operates in both push and pull-based mode. The underlying communication is based on the CoAP protocol. In push-based mode, the component listens to data update requests from things that are pre-configured with the *FnC* endpoint. In pull-based mode, the component actively connects to a list of pre-registered things to collect data from them. In both cases, data exchange format to capture the data from things follows the OBIX message standard.

Listing 1: Sample ALE event: using the extension mechanism within an EPC standard event

```
<reports>
  <report reportName="reportName">
   <group>
```

```
   <groupList>
    <member>
     <epc>urn:epc:id:sgtin:886348213.7508.136164026610</epc>
     <rawHex>urn:epc:raw:96.x304f4d499b57551fb40228f2</rawHex>
     <extension>
      <fieldList>
       <field>
        <name>temperature</name>
        <value>24</value>
        <fieldspec>
         <fieldname>unit</fieldname>
         <datatype>float</datatype>
         <format>degree C</format>
        </fieldspec>
       </field>
      </fieldList>
     </extension>
    </member>
   </groupList>
  </group>
 </report>
</reports>
```

After the data has been collected, the STIS' *FnC* component generates an Application Level Event (ALE) event that will be forwarded to the Capturing Application. The ALE event fully follows the EPCIS specification since it uses the EPCIS extension for storing products' data. Listing 1 shows how the extension incorporates products data into normal ALE events. The sample ALE event holds an extension field called "temperature" with a value of "24" and some other information such as data type or format. This ALE event will be captured into the STIS repository through the Capturing Application along with all other ALE events from RFID readers. Finally, the data is made available to end users or physical devices.

4.2.1 Data acquisition based on OBIX standard and CoAP binding

OBIX attempts to provide a standard XML syntax for representing machine-to-machine (M2M) information provided by embedded sensors and actuators. Its goals are to use enterprise friendly technologies like XML, HTTP and URIs for M2M communication. A goal of OBIX is to provide a standardized representation for common M2M features like data points, histories and alarms extensible for custom enhancements. Although OBIX was designed to work on embedded devices, HTTP and XML are not the best technology choice for Low power and Lossy Networks (LLNs). To address this issue, OBIX specified in the 1.1 specification working draft a custom binary protocol designed for the use in 6LoWPANs. In a recent committee specification draft, a protocol binding to CoAP and message encoding for JSON and EXI have been defined, making OBIX more suitable to the constraints found in low-power and lossy wireless sensor and actuator networks. Since OBIX was designed around the RESTful design paradigm from the very beginning it is a good choice to integrate existing state of the art home and building automation technologies but further allows a deployment directly on the sensor or actuators.

So-called OBIX contracts provide a template for certain sensor and actuator types. The OBIX core specification comes with a set of standard contracts for *data point centric* data representation, *histories*, *alarming* and *watches*.

For the integration with the STIS, the data point centric representation and the history contract are most relevant.

4.2.2 EPC-based device identification

The EPC is designed by GS1 Global as a universal identifier for physical objects. It supports up to 7 different formats for identification keys. An EPC can be represented in a canonical form through a URI. Listing 2 illustrates the syntax and provide an example of an EPC.

Listing 2: Serialized Global Trade Item Number

```
urn:epc:id:sgtin:CompanyPrefix.ItemRefAndIndicator.SerialNumber
urn:epc:id:sgtin:0614141.112345.400
```

To extend the legacy EPCIS to a STIS so that it can embrace things from BAS, things are represented in OBIX objects and EPCs are added the OBIX objects in the form of an Id object. These uniquely assigned EPCs help STIS to easily capture things data into its repository. Listing 3 shows an example of how this technique is implemented.

Listing 3: Adding EPC to OBIX object

```
<obj href="/obix">
    <ref name="about" href="about"/>
    <ref href="/id" is="iot:id"/>
    <ref href="/virtualPresence" is="iot:PresenceSensor"/>
</obj>

<obj>
    <str name="epc" href="epc" val="0057000.123430.2025"/>
</obj>
```

4.2.3 Push-based data acquisition in STIS

In push-based mode, the STIS' FnC component opens a RESTful API, which binds to CoAP protocol, to receive data update requests from things. The API endpoint has to be provided within the sensor or the gateway beforehand. The push-based interaction limits the required communication to the minimum since the sensor can be configured to push new data only if certain events happen.

Listing 4 illustrates the OBIX contract for the interface of a single sensor value at the EPCIS. A basic *real* object is used to collect sensor readings. The current value can be queried with the *val* attribute but more important a standard OBIX history object is provided that allows to push new sensor values using the OBIX *append* operation.

Listing 4: STIS OBIX interface

```
<real href="iot:TemperatureSensor" val="0.0" unit="obix/celsius" is="obix:History"/>
    <int name="count" min="0" val="0"/>
        <abstime name="start" null="true"/>
        <abstime name="end" null="true"/>
        <str name="tz" null="true"/>
        <list name="formats" of="obix:str" null="true"/>
    <op name="query" in="obix:HistoryFilter" out="obix:HistoryQueryOut"/>
    <feed name="feed" in="obix:HistoryFilter" of="obix:HistoryRecord"/>
    <op name="rollup" in="obix:HistoryRollupIn" out="obix:HistoryRollupOut"/>
    <op name="append" in="obix:HistoryAppendIn" out="obix:HistoryAppendOut"/>
</real>
```

4.2.4 Pull-based data acquisition in STIS

In Pull-based mode, the STIS captures things data by polling the sensor or the gateway. It is straightforward in capturing data from a single sensor because there is only one EPC assigned to the sensor. However, in the gateway case, the gateway manages many individual sensors so that the STIS needs to

loop through all of these sensors to capture their data. This can be done via the lobby OBIX object of the gateway, which contains information on all the things the gateway is managing as well as their OBIX interfaces.

This capturing mode imposes the most communication overhead since the server has to actively request for new sensor data. If no new sensor data is available unnecessary traffic occurs. However, this mode provides more freedom to the BAS implementation since little to no effort is required to configure the existing BAS for data uploading.

4.2.5 Gateway or sensor/actuator deployment

The described communication interfaces can be deployed directly on sensor or actuators within constrained lossy and low-power wireless sensor and actuator networks or through a gateway interface for state of the art BAS technologies like KNX, BACnet, ZigBee or EnOcean to mention a few examples. The deployment is most feasible for constrained devices since gateway devices usually have enough memory and computational resources and a good global Internet connectivity, which allows to rely on mature Web service technologies based on HTTP and XML. However, for integration purposes, it is most convenient to have a uniform communication interface. By using CoAP expensive polling requests can be avoided. Furthermore, the usage of CoAP enables the creation of control logic based on IPv6 multicasting as presented in [18].

5 Evaluation

5.1 Case study

This section introduces a realistic, world-wide deployment of the the proposed system which, is used for validation of the work. We deployed STIS at KAIST, Korea while VUT, Austria provides a Building Automation System with oBIX gateways and devices. The purpose of such worldwide deployment is to test the ability to capture/query things' data via CoAP protocol over an IPV6 based Internet. The system is shown in Fig. 3.

5.2 The EPC sensor network at KAIST

The STIS repository at KAIST is based on the open source project Fosstrak [19], which is a state-of-the-art implementation of EPCIS repository. STIS extends the Fosstrak's implementation by adding a RESTful layer and new CoAP-based communication channel. It also features a live feed of event stream that third-party applications can subscribe to receive. The implementation based on Java Servlet technology and is deployed on Apache Tomcat container with a MySQL backend.

STIS FnC uses the open source project Rifidi [20] to capture and filter events from physical things. Rifidi uses an open source Complex Event Processing (CPE) framework called Esper [2] to filter raw data events from physical layer. The raw data are captured into Rifidi through a collection of plugins that connect to things via different communication protocols. We developed two OBIX plugins for the Rifidi server with UDP sockets to support CoAP communication. We use the state-of-the-art CoAP implementation Californium [21] for our CoAP stack due to its usability and maturity. Amongst two OBIX plugins, one acts as a CoAP server to collect thing data in the push-based data acquisition mode. The other acts actively connect to a list of pre-configured OBIX servers to retrieve thing data in pull-based mode.

[2] http://esper.codehaus.org/

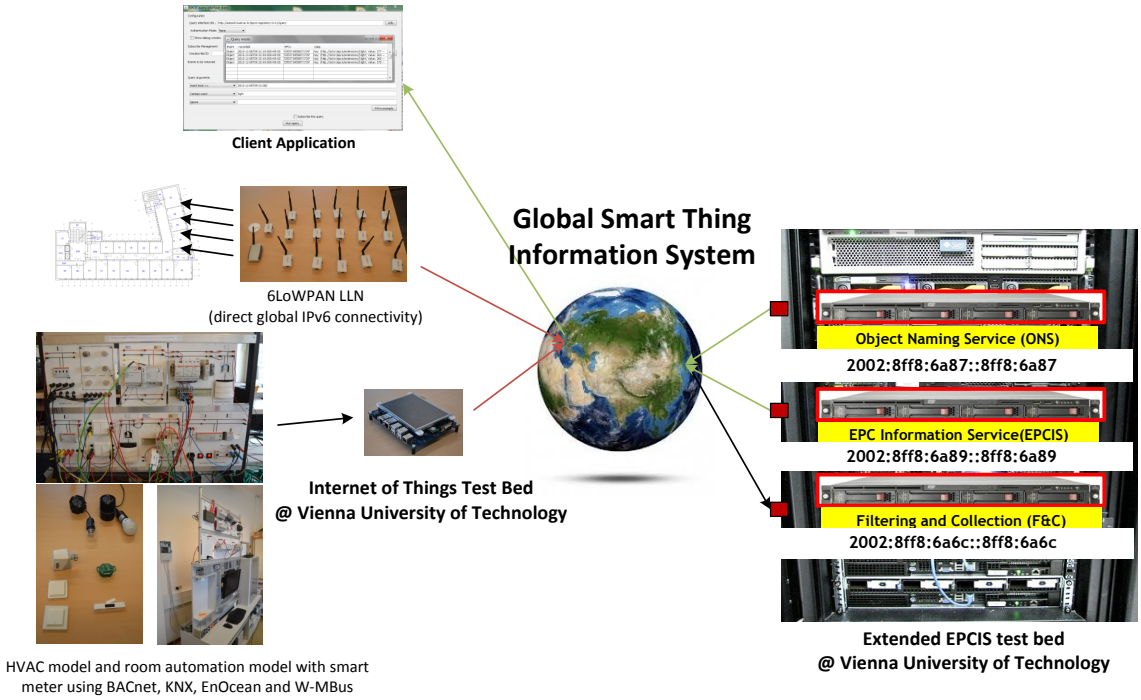

Figure 3: The Deployed System

```
<obj href="/obix">
  <ref name="about" href="about"/>
  <ref href="/id" is="iot:Id"/>
  <ref href="/virtualIndoorBrightnessSensor" is="iot:IndoorBrightnessSensor
  <ref href="/virtualPresence" is="iot:PresenceDetectorSensor"/>
  <ref href="/virtualFanSpeed" is="iot:FanSpeedActuator"/>
</obj>
```

EPC	Data	Action
urn:epc:id:sgtin:1234567.000000.68719476746	presence: false fanSpeed: 0 brightness: 243.0	ADD
urn:epc:id:sgtin:1234567.000000.68719476746	presence: false fanSpeed: 22 brightness: 244.0	ADD
urn:epc:id:sgtin:1234567.000000.68719476746	presence: false fanSpeed: 25 brightness: 237.0	ADD
urn:epc:id:sgtin:1234567.000000.68719476746	presence: false fanSpeed: 21 brightness: 240.0	ADD

Figure 4: Demonstration of Data Capturing from BAS to STIS

5.3 Internet of Things testbed at VUT

An Internet of Things testbed in Vienna is connected to the STIS repository in KAIST using IPv6 connectivity. IoT smart objects equipped with sensors have a direct end-to-end connection with the STIS repository. Contiki based 6LoWPAN devices are used. Therefore, Contiki has been extended to include an OBIX implementation and to offer an according communication interface. For integrating existing sensors and actuators in STIS, a Java based OBIX gateway is used. It can be operated both in a push mode to report latest sensor readings to the STIS or in a pull mode in which the STIS queries the gateway for updated sensor readings. A versatile testbed based on KNX, BACnet and EnOcean devices is

Platform	**STIS**	**NGrinder & NGrinder agent**
Operating System	Ubuntu Linux 12.04 server 64 Bit	Windows 7 64 Bit
CPU	HP ProLiant DL160 Intel Xeon CPU E5504 2.00GHz	Intel i7 870 2.93GHz
Memory	4 GB RAM	8 GB RAM
Software	Apache Tomcat 7.0.54, Oracle JVM 1.7.0_55 64-bits	JVM 1.7.0_40 64

Table 1: Test environment

# of concurrent clients	**# of processes**	**# of threads per each process**
1	1	1
5	1	5
10	1	10
20	2	10
30	2	15
40	2	20
50	2	25
100	4	25

Table 2: Experiment setting for emulation of concurrent clients

used. The gateway implementation based on Java as well as the Contiki based sensor/actuator firmware is provided as open source [22].

In this section, we demonstrated the implemented system by sending and receiving data between KAIST's STIS and VUT's IoT testbed. The demonstration scenario is about a building automation use case in which an administrative application is setup to watch for operations of smart devices in building. The STIS is configured in pull-based mode with the VUT's OBIX gateway address so that its FnC component continuously polls the gateway for data. The administrative application is programmed so that it can detect abnormal data which are collected from devices through the OBIX gateway. When abnormal data are detected, the administrative application can automatically notify the system operators and issue replacement order (based on the device's EPC number) without users intervention. Fig. 4 shows the OBIX gateway's lobby object which lists all the devices being monitored and the Web interface of the STIS where user can see a live stream of data coming in the STIS repository. The live data stream reflects the status of all the devices that are being monitored by the OBIX gateway and gives the administrative application inputs for its decision.

5.4 STIS lightweight interface performance

The STIS is deployed in a server. We emulate concurrent clients to send STIS queries using a Web performance evaluation tool called NGrinder. It can send concurrent requests and measure their response time in an automatic manner. NGrinder can also execute a Java program using its script execution function, so we used NGrinder to execute Californium-based CoAP query client program written in Java. NGrinder emulates each client using a thread. The number of processes and threads per each process for each test case are summarized in Table 2.

Figure 5 based on Table 3 shows how the response time of the STIS CoAP interface and its error rate change according to the number of clients. When only one client sends a CoAP query to the STIS every 2 seconds, the average response time is 214.05 milliseconds and there is no error. This trend continues

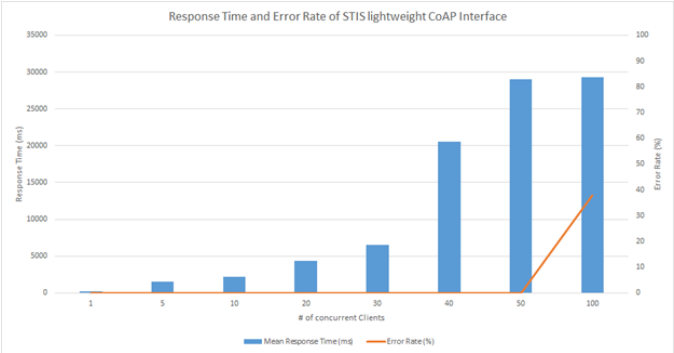

Figure 5: Response time and error rate of STIS lightweight CoAP interface

# of Concurrent Clients	Mean Response Time (ms)	Mean Response Time per # of Clients (ms)	Error Rate (%)
1	214.05	214.05	0
5	1489.92	297.984	0
10	2185.32	218.532	0
20	4364.26	218.213	0
30	6531.12	217.704	0
40	20527.92	513.198	0
50	29001.1	580.022	0
100	29302.2	293.022	37.87

Table 3: Response Time and Error Rate of STIS lightweight CoAP Interface

to the case when the number of clients is 30. It is clearly shown that the mean response time increases linearly according to the number of clients. When it comes to the next case (i.e. 40 clients), the average response time per client increases. Thus, the processing capability of the STIS is gradually saturated, leading some clients to wait more time to get a response. Nevertheless, it is shown that the STIS can process all the queries. However, as for the final case when 100 clients participate in the experiment, the request error rate increases to 37.87%. Please note that the mean response time per clients (i.e. 293.022 ms) is calculated only based on the measured response time of successful requests. To sum up, the SITS lightweight CoAP interface guarantees a comparatively constant and feasible processing capability (around 200 ms) with 30 clients; also STIS can scale up to 50 clients with increased response time.

5.5 Gateway performance

For the scalability analysis of the proposed gateway to integrate building automation systems, a queuing network model of the gateway allows to predict the resource demand and the performance behaviour. Therefore, the different request types that impose load on the gateway are analyzed, measured and used to establish and quantify an analytic model that allows to estimate the scalability of the gateway in certain load scenarios.

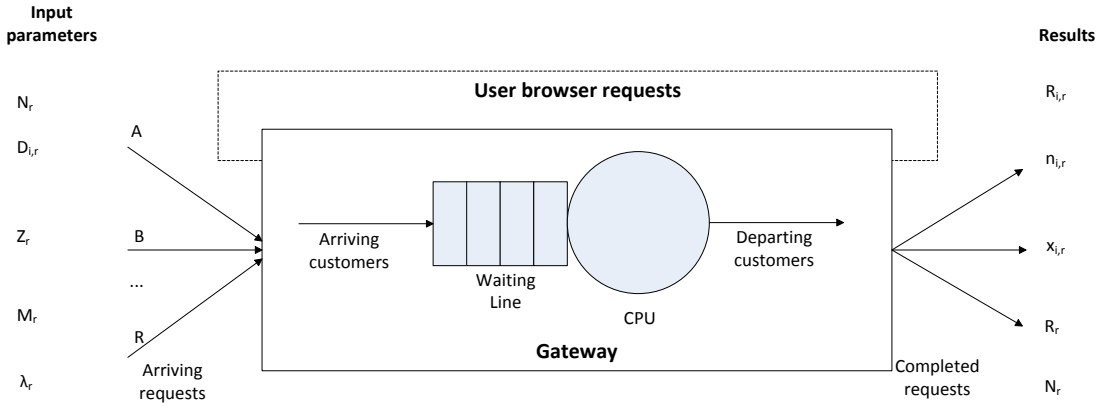

Figure 6: QN model of gateway

5.6 Mixed queuing network model for gateway analysis

A mixed queuing network model is taken to analyze the scalability. This model approach and according algorithms to solve this type of model follow the methodology presented in [23]. For the analysis of the gateway, the various request types are used to define the model. The gateway needs to deal with requests that represent the process data exchange. Group communication used for creating distributed control logic scenarios falls into this category. Also read and write requests on OBIX objects and datapoints for creating application logic can be categorized into this type of traffic. Furthermore, human users may interact with the gateway in order to directly interact with objects or to create further control logic applications.

A mixed queuing network model can be taken to represent these different types. A closed model reflects the interactive user requests that are placed on the gateway. An open model is used to represent related requests to the process data exchange. The resource of interest is the CPU of the gateway.

To solve the combined model of the multi-class open and multi-class closed model, the principle approach is to solve the independent models. In order to represent the influence of the different models on each other, first the utilization of the resources due to the open class model is calculated and depending on the utilization, the service demand of the closed model is elongated. The closed model with the modified service demand is then solved and the average customer queue length of the closed model customers is used for calculating the response times of the open model.

Closed model The closed model is represented through the load intensity vector $\overrightarrow{N}$ and the relevant class descriptor parameters of $(D_{i,r}, M_r, Z_r)$. R is defined with the set of possible request types $R = (WatchService, Read, Write, GroupManagement)$. The *WatchService* customer class represents the recurring requests that occur through the OBIX watch service, issued by a client to update the user interface. In case of the HTTP binding of the gateway, this service can be polled for updates that happened since the last poll. The watch service comes with two different request types. The *pollChanges* operation provides a delta of changes since the last poll and the *pollRefresh* operation provides a complete list of the objects that are currently monitored by the watch. The computational effort strongly depends on the required number of OBIX objects that need to be encoded. The positive effect of the *pollChanges* further depends on the update ratio of the represented objects. If the update ratio is higher than the polling interval, no improvement of *pollChanges* compared to *pollRefresh* can be observed.

Class	Service Demand CPU (ms) Pi	Smart Board	x86 PC
Datapoint Write	298	52	12
Datapoint Read	259	45	1
Object Write	334	51	13
Object Read	265	44	1
Group Communication Write	72	41	16
Bus Event	3	6	1

Table 5: Service demands open model

Table 4 provides the service demands measured on different hardware platforms operating the gateway. For the evaluation a x86 PC platform is compared to a custom developed embedded smart board and the popular Raspberry PI platform. The smart board was developed within the IoT6 research project[3] and is comparable to the Raspberry Pi but comes with a customized extension of transceivers for legacy building automation systems. For acquiring the measurements the operational analysis methodology presented in [24] is used.

Class	Service Demand CPU (ms) Pi	Smart Board	x86 PC
WatchService	159.13	41	1
Write	95.45	32	2
Read	51.54	17.7	7
Group Management	57.71	31	2.5

Table 4: Service demands

Open model For the requests related to process data exchange, an open model is used, since requests may originate within the local network by devices directly sending requests and also local or remote applications that perform control logic scenarios. The open model is represented through R customer classes with a load intensity vector $\overrightarrow{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_R)$ on K devices, where λ_r indicates the arrival rate of class r customers.

Table 5 illustrates the service demands measured at the gateway for the open model.

5.7 Evaluation results

For the evaluation, different scenarios are taken into the account. The main input parameters are the arrival rate of the open class request, the think time and the number of closed class customers. To keep the complexity of the model reasonable the same think time is used for all closed class requests. For the open class requests, the arrival rate is grouped into an arrival rate for process related communication that consists of group communication [18] and bus communication, and further into a group that represents ad-hoc communication generated through external applications which operate through client/server based communication with datapoints and objects. The process communication takes place at run-time without the involvement of any system engineer. This might be traditional non-IP bus communication, as well as IP-based communication using CoAP. The following results show the average response time to be expected for a certain arrival rate and number of clients for the Raspberry Pi platform.

[3] www.iot6.eu

Scenario comparison The achieved gateway performance strongly depends on the pattern and composition of the traffic and the user interaction. Taking the Raspberry Pi platform, the worst case performance can be observed if the process communication is disturbed by high load imposed through ad-hoc communication requests and high user interaction. Reducing the ad-hoc communication to 20% and decreasing the user delay provides a significant performance improvement, but the best performance can be achieved if the ad-hoc communication is limited to 10% and the user interaction is kept at a low interaction rate.

For the comparison of different traffic scenarios, the average response time of the process communication based on CoAP/XML is used.

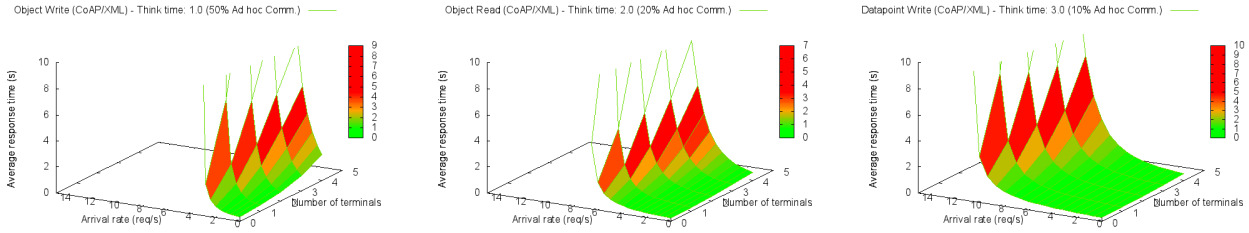

Figure 7: Scenario comparison - Raspberry Pi

Platform comparison The previous results only focused on the Raspberry Pi platform. The results below show the difference between the other hardware platforms that have been considered. A custom developed smart board within the IoT6 research project shows a much better performance compared to the Raspberry Pi platform.

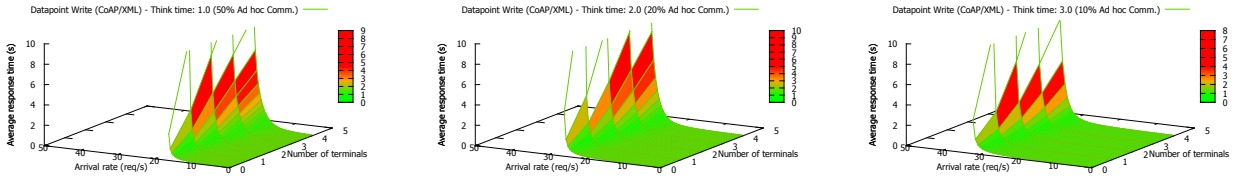

Figure 8: Scenario comparison - custom smart board

However, the performance of an x86 based PC platform is still a magnitude better. The cost difference for the various platforms is also significant.

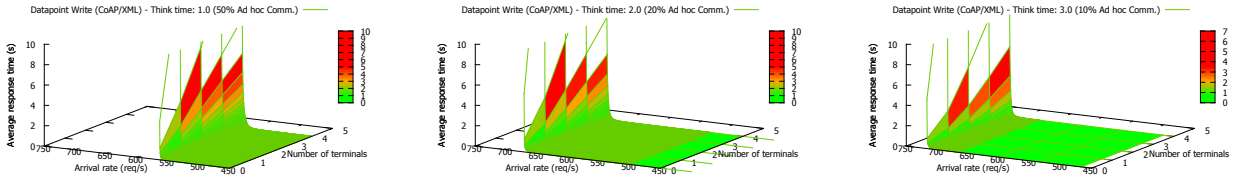

Figure 9: Scenario comparison - X86 platform

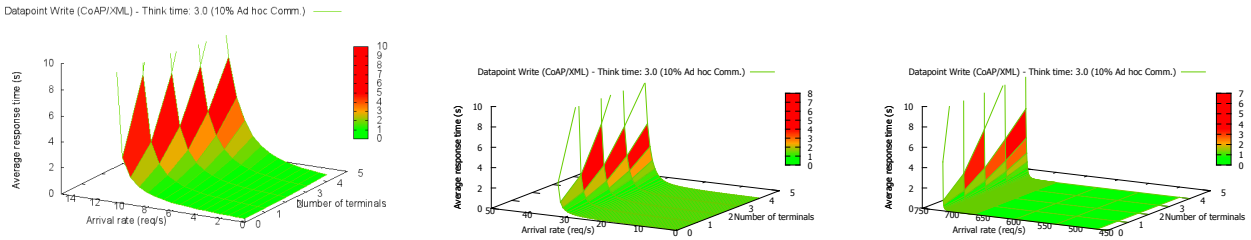

Figure 10: Platform comparison

6 Conclusion

Follow the evolution of the IoT, we showed that traditional information system such as EPCIS can be upgraded and extended to seamlessly embrace new application domains. In this paper we introduce such extension of EPCIS called STIS and an interoperable approach to integrate such information system into an emerging IoT application domains - the Building and Automation System. By leveraging the extension of the EPCIS standard and protocols that suit perfectly to the IoT field (i.e., CoAP/OBIX), the STIS brings a new information infrastructure for IoT data warehousing, specifically BAS data management in this work. We also showed that a lightweight query interface based on REST-ful web service and CoAP protocol is of much benefit to third-party applications. We showcased a realistic, world-wide integration system between our partners that proves its practicability. Further, a performance evaluation of the proposed STIS lightweight interface and a gateway component for building automation systems is conducted to estimate the scalability of the proposed system.

References

[1] GS1, "EPCglobal EPC Tag Data Standard (TDS) v1.6," 2011.

[2] N. K. Giang, S. H. Kim, D. Kim, M. Jung, and W. Kastner, "Extending the EPCIS with Building Automation Systems: a New Information System For the Internet of Things," in *Proc. of the 8th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'14), Bimingham, United Kingdom*. IEEE, July 2014, pp. 364–369.

[3] "Information technology - Home electronic system (HES) architecture - Part 3-10: Wireless short-packet (WSP) protocol optimised for energy harvesting - Architecture and lower layer protocols," IEC 14543-3-10, 2012.

[4] IETF, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," IETF RFC 4944, September 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4944.txt

[5] "BACnet– A Data Communication Protocol for Building Automation and Control Networks," ANSI/ASHRAE Std. 135, 1995-2010.

[6] "KNX Specifications, Version 2.0," Konnex Association, 2009.

[7] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine, 2000.

[8] C. Bormann, A. Castellani, and Z. Shelby, "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, March 2012.

[9] OASIS, "oBIX 1.1 draft committee specification," 2011.

[10] ThingSpeak. [Online]. Available: http://thingspeak.com/

[11] Cosm. [Online]. Available: http://cosm.com/

[12] Sen.se. [Online]. Available: http://open.sen.se/

[13] GS1 Global. [Online]. Available: http://www.gs1.org/

[14] C. Du and S. Han, "Integrating EPCglobal network with Web services," in *Proc. of the 6th International Conference on Service Systems and Service Management (ICSSSM'09), Xiamen, China*. IEEE, June 2009, pp. 329–334.

[15] D. Guinard, M. Mueller, and J. Pasquier-Rocha, "Giving RFID a REST: Building a Web–Enabled EPCIS," in *Proc. of the 2nd International Conference on Internet of Things (IOT'10), Tokyo, Japan*. IEEE, November-December 2010, pp. 1–8.

[16] M. Jung, J. Weidinger, C. Reinisch, W. Kastner, C. Crettaz, A. Olivieri, and Y. Bocchi, "A transparent ipv6 multi–protocol gateway to integrate building automation systems in the internet of things," in *Proc. of the 2-12 IEEE International Conference on Internet of Things (iThings'12), Besancon, France*. IEEE, November 2012, pp. 225–233.

[17] M. Jung, J. Weidinger, W. Kastner, and A. Olivieri, "Building automation and smart cities: An integration approach based on a service–oriented architecture," in *Proc. of the 27th International Conference on Advanced Information Networking and Applications Workshops (WAINA'13), Barcelona, Spain*. IEEE, March 2013, pp. 1361–1367.

[18] M. Jung and W. Kastner, "Efficient group communication based on web services for reliable control in wireless automation," in *Proc. of the 39th Annual Conference of the IEEE Industrial Electronics Society (IECON'13), Vienna, Austria*. IEEE, November 2013, pp. 5716–5722.

[19] Fosstrak. [Online]. Available: http://fosstrak.org/

[20] Rifidi. [Online]. Available: http://rifidi.net/

[21] M. Kovatsch, S. Mayer, and B. Ostermaier, ""moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things"," in *Proc. of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'12), Palermo, Italy*. IEEE, July 2012, pp. 751–756.

[22] M. Jung, J. Chelakal, J. Schober, W. Kastner, L. Zhou, and N. K. Giang, "IoTSyS: an integration middleware for the Internet of Things," in *Proc. of the 4th International Conference on the Internet of Things (IoT'14), Cambridge, Massachusetts, USA*, October 2014.

[23] D. Menascé, V. Almeida, L. Dowdy, and L. Dowdy, *Performance by design: computer capacity planning by example*. Prentice Hall, 2004.

[24] P. J. Denning and J. P. Buzen, "The operational analysis of queuing network models," *ACM Computing Surveys*, vol. 10, no. 3, pp. 225–261, 1978.

Author Biography

Nam Ky Giang received B.Eng degree in Electronic and Telecommunication from Hanoi University of Technology in 2010 and M.S. degree in Computer Science from Korea Advanced Institute of Science and Technology in 2013. Currently he is pursuing a PhD degree at the Department of Electrical and Computer Engineer, University of British Columbia. His research interests lie on the Internet of Things, specifically at the application layer, and IP-based Wireless Sensor Network.

Janggwan Im received B.S. and M.S. degrees in computer science from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 2009 and 2011, respectively. In 2007, he cofounded a startup company named ISEEYOU which developed a web-based user-generated content (UGC) platform and mobile applications, and it was acquired by Kakao Inc. in 2012. His research interests include GS1/EPCglobal standard, sensor networks, Internet of Things (IoT) middleware. He have been an associate director of Auto-ID Lab Korea (www.autoidlabs.org) since 2014.

Daeyoung Kim received B.S. and M.S. degrees in computer science from Pusan National University, Korea, in 1990 and 1992, respectively, and a Ph.D. degree in computer engineering from the University of Florida in August 2001. Since March 2009 he has been an associate professor with the Department of Computer Science, KAIST, Korea. He was an associate professor with the Department of Computer Science and Engineering, Information and Communications University (ICU), Korea, from 2002 to 2009. From September 2001 to January 2002 he was a research assistant professor at Arizona State University. He worked as a research staff member with the Electronics and Telecommunications Research Institute (ETRI), Korea, from January 1992 to August 1997. His research interests include sensor networks, real-time and embedded systems, and robotics. He is a director of Auto-ID Lab Korea (www.autoidlabs.org) and a director of the Global USN National Research Laboratory.

Markus Jung received the Dipl.-Ing. degree in computer science from the Vienna University of Technology, Austria in 2010 and has several years of industry experience in software development related to the banking and financial market sector. Since 2011, he is with the Institute of Computer Aided Automation where he is working as research assistant and pursuing a PhD. Markus Jung is working on research projects in the area of Smart Grids and Internet of Things and focuses on scalable and secure ICT infrastructures with special interests in service-oriented architectures and integration middleware architectures.

Wolfgang Kastner received the Dipl. Ing. and Dr. Techn. degrees in computer science from Vienna University of Technology, Vienna, Austria, in 1992 and 1996, respectively. Since 1992, he has been with the Automation Systems Group, Institute of Computer Aided Automation, Vienna University of Technology, Austria, where he has been holding the position of an Associate Professor for computer engineering since 2001. From 1992 to 1996, he was working on reliable transmission protocols for distributed real-time systems. Since 1997, his research interests have been in the areas of control networks and automation systems with a special focus on home and building automation. His current research targets the field- and management-level integrations of building automation networks, concentrating on the open standards BACnet, LonWorks, KNX, and IEEE 802.15.4/ZigBee. Dr. Kastner is a Founder Member of the IEEE Industrial Electronics Society Technical Committee for Building Automation, Control, and Management.