Integrating IoT Devices into Business Processes

Christian Friedow, Maximilian Völker, and Marcin Hewelt(✉)

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
marcin.hewelt@hpi.de

Abstract. The Internet of Things (IoT) has arrived in everyday life, controlling and measuring everything from assembly lines, through shipping containers to household appliances. Thus, IoT devices are often part of larger and more complex business processes, which might change their course based on events from these devices. However, when developing IoT applications the process perspective is often neglected and coordination of devices is realized in an ad-hoc way using custom scripts. In this paper we propose to employ process model to define the process layer of IoT applications and enact them through a process engine. Our approach thus bridges the gap between physical IoT devices and business processes. The presented implementation shows that those two can be combined without in-depth programming expertise or extensive configuration, without restricting or strongly coupling the components.

Keywords: Business processes · Business event processing
Process automation · Process execution · BPMN · Internet of Things
IoT · Fragment-based case management · Case management · Events

1 Introduction

Business Processes are operated in increasingly complex environments and have to take into account external events that influence the course of the process execution [1]. The complexity is further exacerbated through the rapid growth of devices in the "Internet of Things" (IoT). These devices are used to automate, measure, and control large parts in different environments, starting from industrial facilities up to lighting and radiators in private homes. However, when developing IoT applications, the process perspective is often neglected and devices are coordinated in an ad-hoc way using a different app for each device or custom scripts that realize the integration logic.r Thus, understanding and adapting IoT applications developed this way becomes a burden. There is a mismatch between business processes that include manual tasks, integrate legacy applications, or call webservices, and the ad-hoc logic of the IoT.

Several web-based services like zapier [2] and IFTTT [3] offer an event-based way to integrate different systems and services, including some IoT devices. However, they are limited to simple event-condition-action rules linking a trigger

R. Matulevičius and R. Dijkman (Eds.): CAiSE 2018 Workshops, LNBIP 316, pp. 265–277, 2018.
https://doi.org/10.1007/978-3-319-92898-2_22

event from one system or service to an action in another one [4]. Meyer et al. [5] propose to integrate IoT devices as resources into business processes, but do not address the execution of such processes. Serral et al. [6] suggest a model-driven solution to integrate pervasive services which interact with sensors and actuators. It allows to create context-specific tasks models and execute them in an engine. However, [6] does not consider business tasks, but rather focuses on supporting behavioral patterns of users.

We propose to employ BPMN process models to define the process layer of IoT applications and enact them through a process engine. This extends the framework of [7] with IoT devices. Technically, this contribution provides a way to bidirectionally integrate low-level, physical IoT devices into business processes, in a way that the execution of process instances is influenced by events, e.g. sensor values, and in return, process instances can send commands to those devices. To simplify this connection and to abstract from the concrete physical device, the Bosch IoT Things service is used. By this means, the process engine and the device do not need to share much knowledge about each other and the implementation of each is encapsulated.

The presented implementation is based on several existing systems: the Gryphon process modeling tool, the Unicorn event processing engine, and the Chimera case engine, all introduced in Sect. 2. Section 3 describes in detail how these systems work together to realize IoT applications with a process layer. Afterwards, in Sect. 4, we demonstrate the feasibility of our approach by realizing an usecase that involves several devices, manual tasks, and webservice calls. We summarize and discuss our approach in Sect. 5, pointing out how it could be improved upon in future work.

2 Foundations

The approach presented in this work is built on a few software systems, which will be briefly introduced in the following.

2.1 Bosch IoT Things Service

The Bosch IoT Things service[1] provides an interface for managing so-called “digital twins” in the cloud. For each connected device, e.g. a Raspberry Pi, a digital representation (the “twin”) is stored in the cloud. This counterpart, referred to as “Thing”, consists of several static attributes and features (attributes the value of which change over time), reflecting values of the physical device.

The example in Fig. 1 shows an abbreviated, possible configuration of a Thing, monitoring a truck. A geolocation sensor connected to the device can be represented as a feature comprising properties for longitude and latitude. Now, each time the sensor measures a location change, the device would update the geolocation feature of its digital representation with the new sensor data.

[1] https://www.bosch-iot-suite.com/things/.

```
{"thingId": "truck104",
  "attributes": {
    "no-of-trailers": "2",
    "driver": "51843"},
  "features": {
    "geolocation-sensor": {
      "properties": {
        "latitude": "52.393787",
        "longitude": "13.131836"}},
    "temperature": {
      "properties": {
        "out": "31",
        "in": "24"}}}}
```

Fig. 1. Shortened configuration of an exemplary Thing

Services interested in the device's data can subscribe to changes and get notified each time the digital equivalent is updated. This way, the Bosch IoT Things service abstracts from concrete device particularities and offers a unified interface to access the device's data. The service also offers backwards communication: services can send data and messages back to the device, for example, to give commands. The communication follows a specified format based on JSON, and Things themselves are also represented in this format. To access a Thing, a Rest API, as well as a WebSocket connection can be used.

2.2 Unicorn

Unicorn[2], an event processing platform, was developed within a logistics project for planning for more efficient transport and was presented by Baumgrass et al. [8] and first described in [9]. As an event processing platform, Unicorn gathers events from event producers, processes, e.g. aggregates, filters or enriches them, and distributes them further to event consumers. The processing is done by Esper[3], a Java library based on the event processing language (EPL).

Event producers can publish events to Unicorn using its Rest API, or events can be fetched using so-called event adapters, which actively poll event sources, like web services. Event consumers can subscribe to event queries and are notified by Unicorn each time a relevant event occurred or a query matched, through REST endpoints. In addition, events can be viewed in a provided web-interface.

2.3 Gryphon

Gryphon[4] is a web-based modeler for process models, build on NodeJS and connected to a MongoDB to persist the models. Next to common process mod-

[2] https://bpt.hpi.uni-potsdam.de/UNICORN.
[3] http://www.espertech.com/esper/.
[4] https://github.com/bptlab/gryphon.

els, fragment-based case models can be created, as described by the fragment-based case management (fCM) approach [10]. Additionally, object life cycles can be defined for data objects and their possible transitions used within the case models.

Models can then be transferred directly to connected Chimera instances (see below) for deployment, or exported as JSON to reuse them in other services.

2.4 Chimera

Chimera[5] is a case engine for executing fragment-based case models (fCM). To accomplish that, Chimera takes a fragment-based case model as an input, analyzes it and enables activities, gateways and events based on their data-flow and control-flow dependencies. Running cases and their current state can be viewed in a web-based interface, which also allows for manual execution of activities and data entry.

Important parts of Chimera for the approach presented in this paper are webservice tasks, data-based gateways, receiving events and manual tasks. As the name suggests, webservice tasks are able to call webservices predefined in the model. Data-based gateways are gateways, whose decisions are based on data-objects and their state or their attribute values, and which therefore can be executed (i.e. decided) automatically by the engine. Receiving events are start or intermediate events, that, in order to be executed, register event queries to Unicorn and wait for the fulfillment of these queries. Data from the event notification can be stored in data object. These three model elements are executed without manual intervention, which enables case models that only consist of these types to be executed completely automatically by Chimera.

3 Approach and Implementation

In this section, we present an exemplary approach and implementation to allow physical devices and event processing services to interact with each other. The practical realization is based on the foundations introduced in Sect. 2. A Raspberry Pi is used as a physical device and the Bosch IoT Things service operates the digital twin. The event processing platform Unicorn registers changes of Things and provides events, which are used by the case engine Chimera to start and influence business processes, modeled in Grpyhon beforehand.

In the following, a general, architectural overview is presented, followed by more specific explanations for each component.

3.1 Overview

Figure 2 provides an overview of the overall structure and components involved. The Raspberry Pi communicates with the Bosch IoT Things service, which in

[5] https://bpt.hpi.uni-potsdam.de/Chimera.

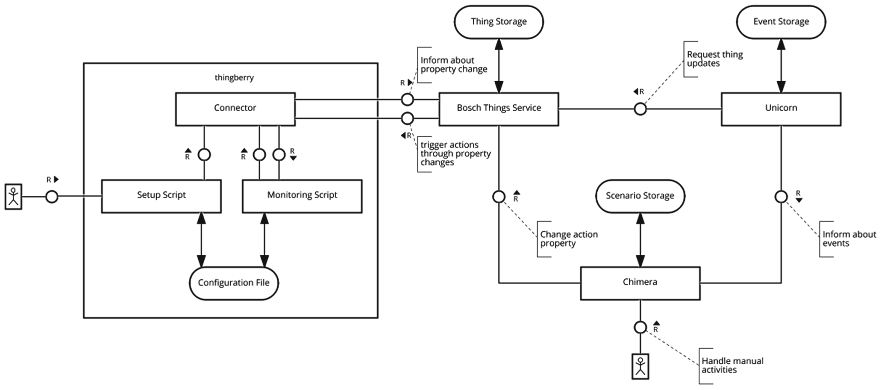


Fig. 2. Fundamental modeling concepts (FMC) model

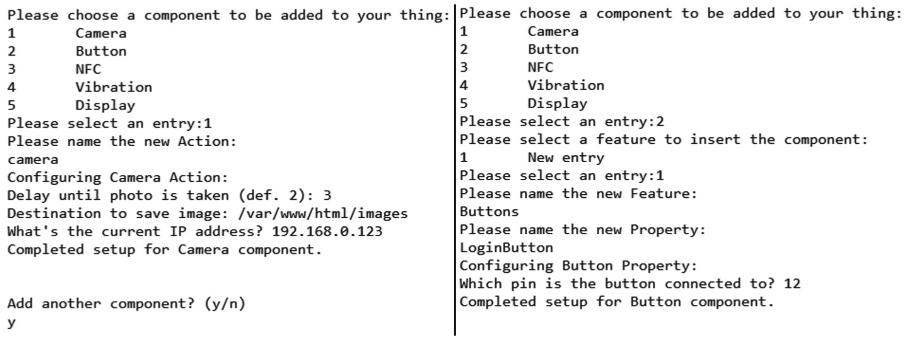
turn is requested by Unicorn for event processing. Chimera, subscribed to Thing related events, gets notified each time a new, relevant event occurs and performs its activities accordingly. In case an action should be performed, Chimera communicates with the Bosch IoT Things service, which then notifies the Raspberry Pi about the change.

3.2 Creating a Digital Representation for an IoT Device

This section describes the process of connecting a physical device to the Bosch IoT Things service and keeping the device synchronized with its digital twin. The thingberry[6] software responsible for this is written in Python and runs on the physical device, in our case a Raspberry Pi. It contains three main software components. First, a setup component that allows to create and store a description of the device, which defines connected sensors, actuators, and attributes. Second, a monitoring component, that observes the connected sensors for value changes. Third, a connector component that, (a) uses the description provided by the setup component to create the digital twin in the Bosch IoT Things service, and (b) connects to the Bosch IoT Things service updating the state of the digital twin, whenever the state of the physical device changes. These components are described in more detail in the following subsections.

Setting up the Representation. The setup component is a command-line tool, used to gather meta information about the device, e.g. the device name, as well as context information, e.g. sensors and the pins they are connected to. An excerpt from an exemplary setup process is given in Fig. 3, which shows the setup for a connected Pi camera and a button. After finishing the setup process, this information is stored locally as a configuration file in JSON format. Thus, configurations can be easily shared and reused for similar device setups.

[6] https://github.com/MaximilianV/thingberry.



```
Please choose a component to be added to your thing:
1       Camera
2       Button
3       NFC
4       Vibration
5       Display
Please select an entry:1
Please name the new Action:
camera
Configuring Camera Action:
Delay until photo is taken (def. 2): 3
Destination to save image: /var/www/html/images
What's the current IP address? 192.168.0.123
Completed setup for Camera component.


Add another component? (y/n)
y
```

```
Please choose a component to be added to your thing:
1       Camera
2       Button
3       NFC
4       Vibration
5       Display
Please select an entry:2
Please select a feature to insert the component:
1       New entry
Please select an entry:1
Please name the new Feature:
Buttons
Please name the new Property:
LoginButton
Configuring Button Property:
Which pin is the button connected to? 12
Completed setup for Button component.
```

Split side by side for clarity.

Fig. 3. Excerpt from an exemplary setup process

In order to reduce the effort and time involved in setting up the digital representation, the user is guided through the process. This eliminates the need to manually write a complex configuration file based on documentation. So far, the setup component offers support for five physical components, commonly used with the Rasperry Pi – buttons, the Pi camera, segment displays, NFC-chips, generic binary components. These can be used in the setup process without any further programming effort or in-depth knowledge about the component. All these components can be configured using the textual interface of the setup component. More components can be added by extending the provided architecture inside the repository, which are also automatically integrated into the setup script.

Two different types of components can be distinguished:

Observers "listen" to changes of the system, like a button press or other sensor values. They are organized in features and properties, and often need to be configured, e.g. which physical pin the button is connected to.

Actions can be triggered externally, e.g. by business process activities. They provide a way to interact with the device or with the environment using the device, like sending a signal or taking a photo. All actions provided by a device, and therefore by the Thing, are grouped as properties within an artificial Thing-feature called "actions" (see the explanation of Thing terminology in Sect. 2.1).

A physical component can be of one or both of the described types. For example, the button component is an observer, which monitors the pin of the (push) button, but the button itself cannot be operated automatically by an action. In contrast, the NFC component serves as an observer, which reads an NFC-chip, as well as an action, which writes to an NFC-chip.

Connecting to the Cloud Service. The connector component takes care of any communication between the physical device and the Bosch IoT Things service. To connect to the service, the provided Rest API[7] is used. Working with the configuration file created by the previous step, the connector component creates a new Thing instance in the Bosch IoT Things service and configures it according to the file. Thing features and properties are instantiated with the provided names and initialized with default values.

During operation of the IoT application, the connector component also updates the digital twin with the latest device state, e.g. sensor values. Due to the fine-grained structure, each change of information can be addressed and processed individually. If a sensor reports a new value, only this specific value can be updated and there is no need to refresh the whole Thing.

Monitoring the Device and Its Components. The third component manages the device at "runtime": Each connected physical component, like a button, is monitored in its own thread, according to the configuration. The decoupled structure of the implementation allows to supervise different components simultaneously, whilst being more error resistant, as each component operates in an own thread. In case a change is registered, e.g. the button is pressed or the measured temperature increases, the new value is assigned to the corresponding feature and property of the Thing. Then, the updated value is provided to the connector component, which updates the digital twin in the Bosch IoT Things service. By comparing the previous value with the updated one, unnecessary calls and updates to the Rest API are avoided.

The monitoring component also manages the return path from the Bosch IoT Things service to the Raspberry Pi and its connected components. On startup, a WebSocket connection to the cloud service is established and within this connection, the script registers for events about changes to the Thing. Thus, every time a Thing changes, an event is received containing information about the affected Thing entity, the changed feature and property, as well as the new value. Since so-called actions are organized in a separate "actions"-feature, action related events can be identified by filtering firstly according to the configured Thing name and secondly, the change must concern an action. Most actions can be triggered by setting the corresponding property-value in the "actions"-features to *true* (i.e. enabling it). Now, each time a property within the "actions"-feature of the current Thing is set to *true*, an event passes the filter and can be processed further. Based on the event's information, the monitoring component determines the requested action and executes it in an additional thread. After completing the action's task, the corresponding property in the twin is reset to allow another execution. A more abstract view is provided in Sect. 3.5 below.

[7] https://things.s-apps.de1.bosch-iot-cloud.com/documentation/rest/.

3.3 Receiving Events from "Digital Twins"

Every time a Thing in the Bosch IoT Things service is updated, an event should be registered inside Unicorn. In order to receive the updates, which we created earlier through the Python script, we developed an event adapter for Unicorn[8]. This event adapter requests the Bosch IoT Things service Rest API for all Things in regular intervals. Afterwards, it calculates the JSON difference between the last two requests according to RFC 6902[9] and converts the updates into events in Unicorn.

Table 1. Mapping from target of change to corresponding event type

Target of change	Event type
Thing was created	ThingAdded
Attribute was changed	AttributeChanged
Feature was changed	FeatureChanged
Property was changed	PropertyChanged
Thing was deleted	ThingRemoved

Events created by this adapter will have different event types based on the target of the change. Table 1 shows the change made to the Thing and the event type of the corresponding event. The events created by the Bosch IoT Things adapter can now be either processed further using EPL or used directly by other services that subscribed accordingly to Unicorn.

3.4 React Flexibly to Events

In order to react to events using fragment based case models, the case model execution engine Chimera is introduced to the workflow. Chimera already implements a configurable connection which connects to the Rest API provided by Unicorn. If a case model containing message receive or send events is deployed to Chimera, the engine will register the specified queries in Unicorn. Message receive events will additionally register a callback to Chimera, which is called by Unicorn each time the registered EPL query is matched. The implementation of the connection between Unicorn and Chimera has been described in [11] and conceptually extended in [7].

Bringing together the events created in Unicorn when the digital twin is changed and a case model using the mentioned message receive events, opens up the possibility to drive a case model using events from IoT devices. That enables case model execution engines like Chimera to automatically decide gateways or

[8] https://github.com/bptlab/Unicorn/tree/dev/EapEventProcessing/src/main/java/de/hpi/unicorn/adapter/BoschIot.

[9] https://tools.ietf.org/html/rfc6902.

change the state of objects, based on sensor values or real life events measured by IoT devices. The next challenge was to trigger actions, like taking a photo or displaying some text on a IoT device, using certain activities inside the case model.

3.5 Sending Commands from Business Activities

As mentioned before, the communication between cloud service and device is bi-directional. Thereby, e.g. business process instances can trigger actions on physical devices and affect the environment. In Sect. 3.2, "actions" were introduced and how they can be triggered on devices. To accomplish the required property-change, so-called webservice-activities are used inside Chimera. Those activities can be enriched with a URL that is called as soon as the activity is executed. In this case, the webservice-activities are configured to perform a request to the Bosch IoT Things service Rest API, which changes the desired property in the digital twin.

> Example: A business process requires a device (the Raspberry Pi) to take a photo. During the modeling process, a webservice-activity is inserted at the desired point and configured with a request to enable the "camera"-property in the "actions"-feature on the correct Thing. The Raspberry Pi configured for this Thing, now receives an event for this change through its WebSocket connection to the Bosch IoT Things service. It determines that the "camera"-property was connected to the configured Pi-Camera-component during the setup process and executes the logic behind it (taking a photo). As it is very likely that the image needs to be viewed at some point later, the component also updates the "camera"-feature with the path of the last photo taken. In addition, the value of the "camera"-property in the "action"-feature is reset (i.e. disabled) and therefore ready to be triggered again.

4 Evaluation

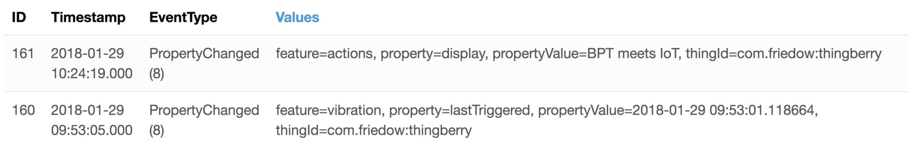
In order to evaluate the approach presented above to connect Internet of Things devices to business processes, we realize an exemplary use case. The use case should not only combine different physical components or only be of theoretical or abstract use, but should also demonstrate, how the combination of IoT and business processes can simplify and enhance our everyday lives.

The Idea is a simple coffee machine billing system. The system aims to automate the process of counting the coffee amount for each user, as well as detecting potential coffee theft, to simplify the billing for a shared coffee machine. In a successful execution of the process, a user should be identified before using the coffee grinder and the amount of consumed coffee in the corresponding user account should increase. If the coffee grinder is used without previous identification the action should be treated as theft and should be documented.

Starting from this informal description, we determined the sensors and actors necessary to implement the use case. In order to ease the authentication process for users, we connected an NFC sensor module to a Raspberry Pi. We also connected a vibration sensor attached to the coffee grinder, in order to detect when it is used. Finally, we connected a display and a camera to the Raspberry Pi, to visualize process progress and take photos of potential thieves.

Creating a Digital Representation for the Raspberry Pi. After connecting all sensors and actors the Raspberry Pi, a digital representation needs to be created. We implemented the scripts for all components in Python and executed the setup script, introduced in Sect. 3.2, which initialized the digital representation of the Raspberry Pi, afterwards. As the initialization finished, the Raspberry Pi was ready to be monitored by the daemon script, which pushes changes of sensor values and receives execution commands for actions.

Receiving Events from Digital Twins. The Thing inside the Bosch IoT Things service already receives updates from the Raspberry Pi and to get these updates into Unicorn too, the Bosch IoT event adapter inside an existing Unicorn instance needs to be started. No additional customization or configuration was required at this step.



ID	Timestamp	EventType	Values
161	2018-01-29 10:24:19.000	PropertyChanged (8)	feature=actions, property=display, propertyValue=BPT meets IoT, thingId=com.friedow:thingberry
160	2018-01-29 09:53:05.000	PropertyChanged (8)	feature=vibration, property=lastTriggered, propertyValue=2018-01-29 09:53:01.118664, thingId=com.friedow:thingberry

Fig. 4. Events concerning the Raspberry Pi

Figure 4 shows a vibration event that was successfully pushed to Unicorn from the the Raspberry Pi, as well as an action event which will show a message on the display.

React Flexibly to Events. Now that the updates of sensor values are available in Unicorn, they can be used by other systems like Chimera. But before actually using Chimera to receive and work on the basis of events, we had to model our exemplary use case as a fragment-based case model. Therefore, we captured our example in eight fragments, modeled them in Gryphon and deployed the case model to Chimera. Figure 5 shows the case fragment in which a user is able to use the coffee grinder if he is currently authenticated.

In order to receive event notifications from Unicorn, Chimera registers EPL queries for each event-receiving model element. From this point on, every time Unicorn receives an event which matches the query, Chimera is notified and the process model is powered by this event.

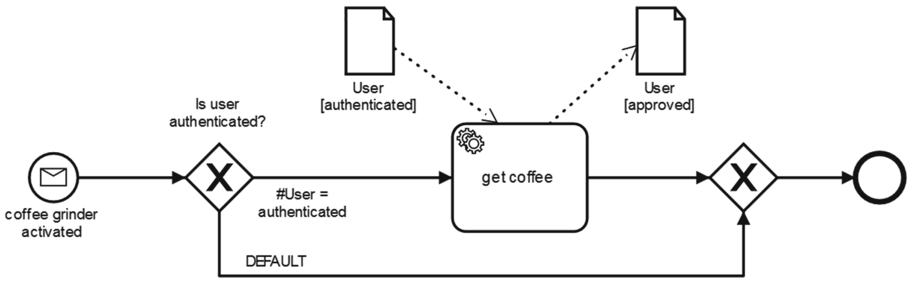


Fig. 5. Exemplary case fragment "Get Coffee"

Sending Commands from Business Activities. After the exemplary showcase successfully received events and powered fragment based case model instances with it, the last task was to trigger actions like taking a photo or showing some text on the Raspberry Pi's display. As already described in Sect. 3.5, actions are triggered by setting certain properties on the Thing inside the Bosch IoT Cloud. To achieve this, we defined webservice tasks inside the case model, changing the corresponding property. These can then be automatically executed by Chimera, if they were enabled. That opened up the possibility to take a photo or display some text using just the executed case model itself.

Wrap-Up. The small use case clearly demonstrates the level of abstraction and the capabilities of the underlying implementation. Whereas the Raspberry Pi just sends its data to the Bosch IoT Things service, having no knowledge about other connected systems, Chimera and Unicorn do not need any detailed information about the device and its physical characteristics. Therefore, it furthermore shows the power of the connection between Internet of Things and business process management and how manual tasks can be automated.

5 Conclusion

The approach presented in this contribution allows to coordinate the devices used in an IoT application using a process engine for the process logic. It can also be used to extend existing business process or case models by integrating external events produced by IoT devices. Thus, with our approach, business processes can make use of real-world sensor data, while on the other hand changing the physical state of the world, by triggering actions. By using a defined and documented interface, the physical world and its model representation can be kept decoupled, allowing to reuse device data in various instances and to access data from multiple different devices from within a single instance.

The presented implementation provides an uncomplicated way to connect small and inexpensive devices with business processes via a cloud service. For the implementation we combined several existing systems. While previous work existed for the connection of Unicorn and Chimera [7,11], the connections between Unicorn and the Bosch IoT Things service, as well as the Bosch IoT Things service and the IoT device originate from this contribution.

One limiting factor of our approach are the sensor and actuator components connected to the Rasberry Pi. For each of these components custom code has to be written to read sensor values or trigger actions. Our implementation already supports several sensors and actuators, like buttons, binary sensors, the camera module, the NFC reader, and a display. It provides also templates that can be sub-classed to support further components, thus reducing the programmatic effort to realize future use cases.

Further the specification of webservice tasks in the case models requires a lot of knowledge about the setup; concrete names of Things, as well as their feature and property names must be available at modeling time and the corresponding Rest API call needs to be assembled manually. In future work we want to examine, how this step can be made more flexible by allowing to define the webservice tasks at deploy or run time. This kind of flexible late binding has already been implemented for email tasks which can be configured on a per-case basis at runtime. Another approach would be to specify Things as well as their features and properties as part of the data model in Gryphon. This is already done for event types that are modeled in Gryphon and registered with Unicorn when the case model is deployed.

References

1. Baumgraß, A., Botezatu, M., Di Ciccio, C., Dijkman, R., Grefen, P., Hewelt, M., Mendling, J., Meyer, A., Pourmirza, S., Völzer, H.: Towards a methodology for the engineering of event-driven process applications. In: Reichert, M., Reijers, H.A. (eds.) BPM 2015. LNBIP, vol. 256, pp. 501–514. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42887-1_40
2. Zapier: Zapier documentation (2018). https://zapier.com/developer/documentation/v2/. Accessed Mar 2018
3. IFTTT: IFTTT documentation (2018). https://platform.ifttt.com/docs. Accessed Mar 2018
4. Rahmati, A., Fernandes, E., Jung, J., Prakash, A.: IFTTT vs. Zapier: a comparative study of trigger-action programming frameworks. CoRR **abs/1709.02788** (2017)
5. Meyer, S., Ruppen, A., Magerkurth, C.: Internet of Things-aware process modeling: integrating iot devices as business process resources. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 84–98. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38709-8_6
6. Serral, E., Valderas, P., Pelechano, V.: Context-adaptive coordination of pervasive services by interpreting models during runtime. Comput. J. **56**(1), 87–114 (2013)
7. Mandal, S., Hewelt, M., Weske, M.: A Framework for Integrating Real-World Events and Business Processes in an IoT Environment. In: Panetto, H., et al. (eds.) On the Move to Meaningful Internet Systems. OTM 2017 Conferences, OTM 2017. LNCS, vol. 10573, pp. 194–212. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69462-7_13

8. Baumgrass, A., Di Ciccio, C., Dijkman, R.M., Hewelt, M., Mendling, J., Meyer, A., Pourmirza, S., Weske, M., Wong, T.Y.: GET Controller and UNICORN: event-driven process execution and monitoring in logistics. In: BPM Demo Session, pp. 75–79 (2015)
9. Herzberg, N., Meyer, A., Weske, M.: An event processing platform for business process management. In: EDOC. IEEE (2013)
10. Hewelt, M., Weske, M.: A hybrid approach for flexible case modeling and execution. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNBIP, vol. 260, pp. 38–54. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45468-9_3
11. Beyer, J., Kuhn, P., Hewelt, M., Mandal, S., Weske, M.: Unicorn meets Chimera: integrating external events into case management. In: BPM Demo Track, vol. 1789, CEUR-WS (2016)