Shared Internet-of-Things Infrastructure Platform (SIoTIP)

Part 1: Application case description

This project involves the development of a platform for applications to manage and automate a shared Cyber-Physical System (CPS) infrastructure. This document describes the project and the application domain, and outlines the existing technological and business constraints.

Section 1 provides some background about the domain and introduces the envisioned platform. Subsequently, Section 2 elaborates further on the main stakeholders involved in our system. This is followed by Section 3, which zooms in on further details of the system under design.

1 Context and Summary

Building automation technologies support the management and control of buildings and physical infrastructures. Their adoption is driven by convenience, comfort, energy efficiency, security, and business value. Applications include centralized control of lighting, HVAC (heating, ventilation, and air conditioning), management of appliances, security locks of gates and doors, and other systems.

The popularity of such automation systems has, both in industrial and domestic contexts, increased significantly in recent years due to improved affordability of hardware and increased inter-connectivity via mobile networks and devices such as smartphones and tablets.

This project involves the development of a platform for applications to manage and automate a shared Cyber-Physical System (CPS) infrastructure.

Figure 1 provides a schematic overview of the set-up of the envisioned platform. It illustrates a number of Internet-of-Things (IoT) devices –**sensors** and **actuators**– that are physically plugged into so-called **motes**, installed in two distinct buildings. These devices connect to **gateways**, which in turn exchange information with an online back-end, the **Online Service**.

The main function of the envisioned platform is to support the development, delivery, and management of **applications** that interact with the available infrastructure made up of sensors and actuators. These applications enable monitoring (part of) a building, and may range from being very generic e.g. controlling the temperature in a room, to highly specialized, e.g. steering and monitoring production processes.

The building control system described above will be deployed on top of a **shared** IoT platform. A main advantage of adopting a shared infrastructure approach is the possibility to integrate technologies which are commonly thought of as separate systems, such as fire control and heating systems. As such, costly duplication of separate sensors with similar capabilities in single building can be avoided (no siloed systems).

Positioning

In this assignment, we assume the point of view of a company that develops the software for the described shared IoT platform, more specifically the platform software for the gateways and the online back-end. We primarily target industrial settings, ranging from office buildings to manufacturing companies with geographically distributed plants, in which business value can be increased by more closely monitoring and controlling business assets or (further) automating their business processes.

For example, the temperature and humidity in server rooms can be automatically monitored and adjusted, or physical access to protected areas can be controlled. As another example, companies could automate their production process, thereby avoiding the need for dedicated, more expensive Industrial Control Systems (ICS). In the future, other business applications are likely. Furthermore, the domestic market, i.e. home automation, is continuously growing thereby providing extra revenue opportunities.

Business model. The shared CPS platform that is developed will be named "**SIoTIP**" (acronym for Shared Internet of Things Infrastructure Platform), and our company "the SIoTIP Corporation". Our

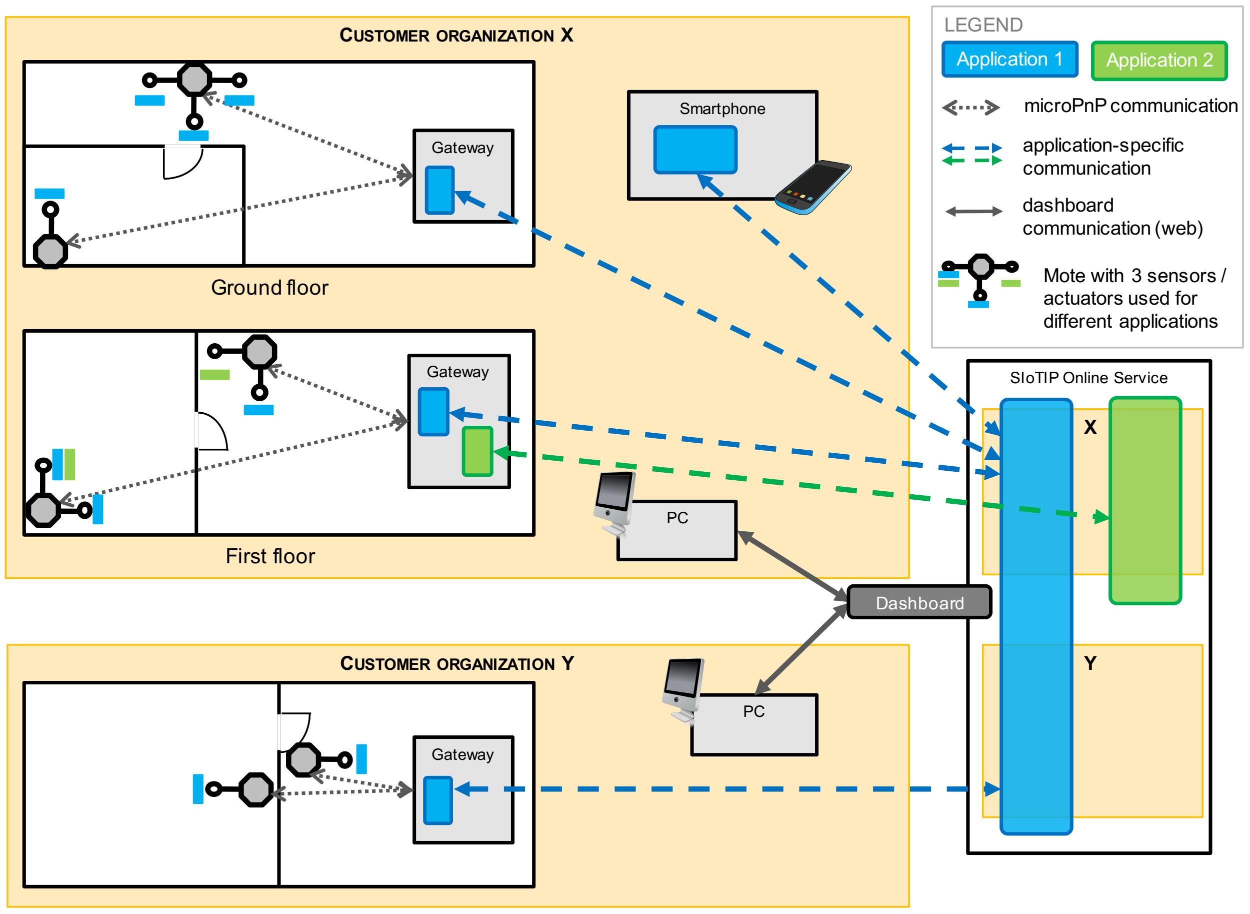


Figure 1: High-level overview of a building control system installation.

goal is to sell SIoTIP gateways and additional hardware devices, such as sensors, to deploy a (shared) IoT infrastructure on top of which third party applications can be offered. Once a shared infrastructure is broadly available, application developers will be incentivized to use our platform for delivering their services (instead of selling their own dedicated hardware and software).

We will operate a service model, where the main revenue is expected to be generated by the services provided via our online back-end (called "**Online Service**" in the remainder of this description). The actual services provided are twofold.

On the one hand, companies want to increase efficiency through automation and simplification of building management using IoT devices. We sell them all the required hardware (i.e. gateways, motes, sensors, and actuators) to easily set up their own IoT infrastructure. Furthermore, we provide them with an inexpensive, flexible and reliable platform via which they can utilize their installed infrastructure through a range of applications.

On the other hand, application providers want to be able to develop and sell applications leveraging the available infrastructure to these companies in a cost-effective manner. Since such applications are often domain-specific and highly specialized, they are best developed by domain experts. Therefore, we offer these third-party developers a reliable, robust platform on top of which a wide range of applications can easily be built, tested, updated and delivered. Furthermore, the platform allows application providers and potential buyers to meet at a low entry cost.

Over time our platform will collect significant amounts of data: raw data from sensors as well as application (usage) information. This data trove can be utilized to improve our provided services and provide feedback to application developers, but also provides big data analysis opportunities we can offer to other (new) parties such as researchers or governments.

Initially SIoTIP is oriented towards the Belgian market, aiming to provide 50 different applications, and reach an install-base of around 200 organizational customers by the end of the first business year. In the longer term, the ambition is to grow and provide our services across the globe.

2 Main stakeholders

The **Internet of Things** paradigm allows for complex system compositions involving multiple stakeholders. The interests of all involved stakeholders, e.g. those installing the IoT devices and those developing applications on top of the platform, have to be taken into account. This section provides an overview of the main stakeholders and summarizes their points of interest.

Customer organizations. The customer organizations' main interest is that they want to subscribe to applications in a hassle-free manner. For example, if new hardware, e.g. extra sensors, is required, the purchase and installation of this hardware should happen as transparently as possible from the point of view of the customer organization. Moreover, applications should be automatically activated if all their requirements are satisfied without requiring any further interactions from the customer organization. Similarly they want to be able to unsubscribe from applications they no longer need or use. Finally, customer organizations should be able to get an overview of the invoices for their application subscriptions. Examples of customer organizations are the fire department and security departments of an airport, or an operator controlling a plant or subsystem in an industrial context.

Infrastructure owner. This stakeholder owns and manages the physical infrastructure, e.g. the building(s), as well as the gateways, motes, sensors and actuators installed in it. Infrastructure owners want to be able to buy new hardware, e.g. to add new sensors or replace broken ones. Note that installing new sensors or actuators can be triggered by customer organizations that want to subscribe to a new application or as a result of updates to an already-deployed application. Owners also want to be capable of monitoring whether all their connected devices are working correctly. Thus, they want to be notified in case their installed hardware fails, e.g. if one or more sensors stop sending data.

Infrastructure owners also want to allocate their installed sensors and actuators to specific customer organizations. For example, an infrastructure owner renting office space in a building to several companies can decide that each company (i.e. customer organization from the point of view of SIoTIP) can only use sensors and actuators installed in their respective offices and those installed in common areas such as hallways. Note that although sensors and actuators can be shared between multiple customer organizations, each piece of hardware has exactly one owner.

Furthermore, infrastructure owners are also responsible for maintaining a topology of their infrastructure. A topology is a virtual view of the infrastructure that documents where devices are installed. In general, sensors and actuators can be assumed to be physically in the same location as the mote they are attached to. However, in the future it may be possible to hook up some motes to nearby devices via a wire, e.g. for door/window sensors.

Additionally, a topology documents any other relevant relations between devices. For instance, a topology can contain that two sensors are equivalent or complementary for a specific purpose. For example, an infrared sensor and microphone sensor installed in the same room can both be used to detect presence of people in this room. The possible relations are typically application-specific and thus likely to change over time.

The information in the topology can be essential for the correct functioning of applications. For example, an access control application must unlock the correct doors and a fire detection application must activate the correct sprinklers in case of fire.

Furthermore, the information in the topology also allows SIoTIP to provide some reliability. Knowledge of the position of the sensors and actuators can, for example, be used to compare a sensor reading to values of nearby, similar sensors to detect anomalies and/or faulty hardware and notify the appropriate stakeholder. Moreover, in case of device failure applications can leverage the topology to automatically switch to a nearby and equivalent device.

End-users. These stakeholders actively interact with the deployed application instances. Such interactions can, depending on the application, be performed via a plethora of devices, e.g. a mobile app on a smartphone or a web browser on a personal computer. For example, a security guard may provide intermediate reports at certain checkpoints on his or her round using a mobile app. For all end-users, the use of applications should simplify or automate (parts of) their daily tasks, while the presence of sensors and actuators should not hinder them.

People who interact with SIoTIP by influencing sensors only passively or unknowingly use the applications and are not considered to be end-users, e.g. visitors or burglars.

Device manufacturers. These stakeholders manufacture and sell the sensors, actuators, motes, and other peripherals used in SIoTIP.

Application providers. The main goal of these stakeholders is to sell subscriptions to their applications running on top of SIoTIP to customer organizations. These providers often have expertise in a specific domain, e.g. building-level physical security, and thus develop specialized applications for that domain. Therefore, they require a development environment providing access to the APIs provided by SIoTIP (e.g. to facilitate communication between the different parts of SIoTIP) as well as the possibility to test and debug applications. This allows them to focus solely on the development and maintenance of their applications and not on the underlying hardware such as gateways or specific sensors, or the exact communication protocols.

Realistically, it must be possible to update existing applications in a smooth and user-friendly manner, for example, to add new features to an application or to address bugs. From the point of view of the application provider, this should not be more complex than adding a new application. Thus, SIoTIP has to support updating running applications without requiring intervention from the application provider.

SIoTIP system administrators. The system administrators are responsible for monitoring and maintaining SIoTIP for its correct working. This includes, among others, the performance and scalability of the Online Service, but also monitoring the workings of the individual gateways. They are also responsible for assisting other parties (such as infrastructure owners) in case they have problems with SIoTIP. They are notified when alarming events threaten the platform, e.g. failure of internal components or a failure to communicate with one or more gateways.

The SIoTIP Corporation. This stakeholder wants SIoTIP to be a successful platform for deploying IoT devices and applications. Hence, they want a large number of infrastructure owners to make their infrastructure available for customer organizations. The majority of revenue will be created by customer organizations that subscribe to applications on SIoTIP. To accomplish this, attracting application providers to develop high-quality applications for the platform is essential.

Telecom operators. Telecom operators provide means for the gateways to communicate with the Online Service and vice versa. A service level agreement between our company and the telecom operators determines the capabilities of these communication channels.

Note that a CPS does not necessarily has to be connected to the Internet, but it could be a secluded system (e.g. physically confined to an airport building). In practice, however, the connection between gateways and the online back-end will commonly use regular internet links, and the IoT devices will use standardized Internet technologies such as IPv6.

Server providers. This stakeholder provides physical or virtual servers that will be used to deploy and operate the Online Service. A service level agreement between our company and the server providers determines the server up-time constraints and the reaction time in case of hardware problems.

3 Details of SIoTIP

As depicted in Figure 1, SIoTIP consists of two distinct but complementing parts: gateways and the Online Service. Gateways are installed in the physical environment (a building) of an infrstructure owner, each of them associated with their own set of motes, hence sensors and actuators. Different floors or local sites of a large plant can each have their own gateway.

The Online Service is an online back-end that serves the purposes of data storage and synchronization, application management, and offers a number of management and control dashboards to the involved stakeholders. The Online Service has no practical limitations in terms of processing power, memory, and storage resources.

The gateway has direct access to the sensors and actuators attached to its connected motes, while the Online Service can only access these devices indirectly via the gateways.

Application support is one of the primary functionalities that SIoTIP should provide. Applications are distributed, consisting of parts running on the Online Service, and optionally on local gateways and mobile devices.

3.1 IoT devices and technologies

There are many competing IoT technologies in the market. In theory, the SIoTIP system could be made compatible with any IoT technology stack. Initially (and for this assignment) we will use a technology called MicroPnP[1].

The following device types will be used (technical details can be found in Appendix A):

Sensor is a hardware device that is physically plugged into a mote (see Figure 2). Sensors produce measurements and send them, via their mote, to the gateway. For example, a temperature sensor would measure the current temperature of the room it is located in.

Actuator is a hardware device that has one or more actions associated with it. For example, a switch can "turn on" and "turn off". Similar to sensors, actuators are also physically plugged into the MicroPnP motes.

MicroPnP mote (or just mote) is a device that can host sensors and actuators, and connects these to a mesh network. As illustrated in Figure 2, motes are powered either by a battery or they can be plugged into an electrical or light socket. Each MicroPnP mote has three USB-like peripheral connectors into which sensors and actuators can be hot-plugged.

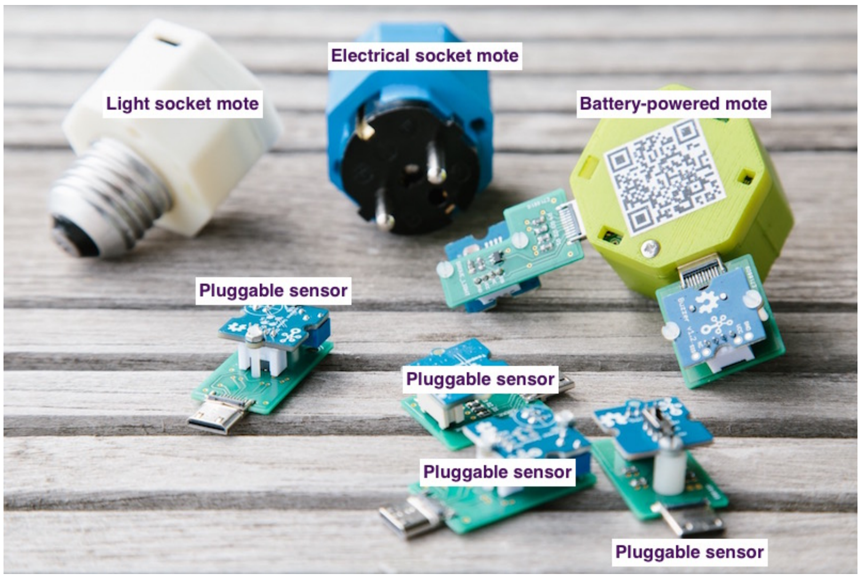


Figure 2: MicroPnP motes and sensors

MicroPnP network manager is a physical device which offers connectivity to the mesh network used by the motes and which can also be connected to a secondary network (e.g. a wired network). It runs an embedded operating system and can be used to run custom programs (e.g. written in Java). However, resources in terms of CPU, memory, and storage are fairly limited. By default, it contains software to access the motes and reliably send and receive messages via the mesh network. Hence, via the software on this device, the motes, sensors, and actuators can be accessed. For connectivity, the 6LoWPAN[2] technology is used. Pairing motes with network managers is done out-of-band using a cable to exchange network information.

SIoTIP gateway is a MicroPnP network manager that runs additional SIoTIP software on top of its embedded operating system. This software relays the messages from connected devices to the Online Service and vice versa, manages the sensors and actuators, and can run some application logic. Due to the limited resources available on a network manager, it can only run lightweight applications.

[1]The MicroPnP technology was originally developed at imec-DistriNet, and is now marketed by the KU Leuven spin-off VersaSense.

[2]`https://datatracker.ietf.org/wg/6lowpan/charter/`

3.1.1 Device management

As the system is a platform for a shared CPS, it should have the ability to interact with and manage the connected IoT devices. Ease of installation is an important concern in this context: new sensors, actuators, and motes should be registered to the gateway automatically, and made available to applications. The plug-and-play functionality of the MicroPnP devices helps in achieving this. Each device is initialized with a functional, default configuration, which can then be adjusted by the platform or application instances (e.g. setting the sampling rate of a sensor).

3.1.2 Technology stack

For resource access, the MicroPnP devices use interfaces compliant with the Constrained Application Protocol[3] (CoAP), which is a RESTful protocol. The motes offer their attached devices, i.e. sensors and actuators, as CoAP resources that are available via GET/PUT. The content (e.g. the value provided by a temperature sensor) is encapsulated into a JSON message. The different types of messages a mote can send the SIoTIP gateway are summarized below, the exact interfaces are detailed in Appendix C.1.

1. **Data** Motes use this to send information from one of the connected sensors or actuators to the gateway (e.g. a temperature reading, buzzer is on, motion is detected). The message also contains an explicit "sensor/actuator type" indicating how the contained value must be interpreted and a timestamp. For example, a data message could be indicated to come from a temperature sensor measuring in degrees Celsius, allowing to correctly use the contained value and, for instance, not as degrees Fahrenheit.

2. **Mote heartbeat** Every mote periodically sends a heartbeat to indicate it is alive. This includes a list of the sensors and actuators connected to it and a timestamp indicating when the heartbeat was generated.

3. **Device plugged in/removed** Whenever a new sensor or actuator is inserted in the mote or removed from it, the mote immediately sends a notification to the gateway. This message specifies the sensor/actuator type of the inserted/removed device and a timestamp.

All sensors will relay new information to the gateway via their mote. This can be done periodically, e.g. for temperature readings, or triggered by state changes, e.g. detection of motion. In case of periodic communication, the sampling rate of the sensor can be configured and this rate will depend heavily on the nature of application instances that depend on that sensor. Request-response based retrieval of information is also possible but is, with the currently supported devices, only used for configuration (e.g. requesting the current sampling rate of a sensor).

Relaying information to the motes by the gateway is done by GET and PUT requests. The former can be used to retrieve a list of attached devices or specific information about a specific sensor, actuator, or mote (e.g. a configuration parameter such as the sampling frequency). The latter is used to update specifications (e.g. set the sampling rate of a sensor) and to send actuation commands. The corresponding operations are detailed in Appendices C.2 and C.3.

Since IoT is regularly applied to new domains and due to the constant technological evolution, it is very likely that over time new (types of) sensors and actuators have to be added. Also, integration with specialized sensors/actuators for domain-specific applications is possible. An example are sensors to detect water leaks and actuators to control valves in a water leak-detection application. In these cases, a new sensor/actuator type will be created and SIoTIP will need information on how to interact with them (e.g. writing value "1" to open a valve). Furthermore, it would be possible to integrate technologies other than MicroPnP provided by VersaSense in the future.

Note that MicroPnP motes are constrained by a trade-off between latency, energy, and bandwidth. Motes can be battery-operated, which means that they sleep most of the time to save energy. The lower the latency of the motes, the higher the energy consumption. Lowering the latency is also tied to a lowered bandwidth. Furthermore, bandwidth in the mesh network is currently limited to roughly 26 data packets per second. This makes it currently impossible to support features such as live video streams using the MicroPnP technology.

3.2 Online Service

The main functions of the Online Service are discussed below.

[3] `https://tools.ietf.org/html/rfc7252`

3.2.1 Gateway - Online Service connection.

Gateways establish a connection with the Online Service which remains open. Gateway installation involves connecting it to the internet via a secondary network interface (i.e. a wired or WiFi network). The gateway will then register itself with the Online Service. The gateway identifies itself using a gateway identifier which is used by the system to link it to the infrastructure owner that has originally purchased it (information which was registered on purchase). Afterwards, communication in both directions is possible, as described below.

A gateway should periodically synchronize data, such as sensor measurements, with the Online Service. The synchronization interval depends on the applications the respective customer organizations are subscribed to. It is essential that the synchronization protocol works correctly in the presence of non-reliable network communication, i.e. that there is no loss of data or inconsistencies. Besides periodic synchronizations, important data such as alarms and notifications must be communicated to the Online Service as soon as possible.

The Online Service also has to communicate with the gateway to, for example, update sensor configurations or activate actuators. This is done as soon as the corresponding command is generated, which can be triggered by end-user input via an application front-end, activation of a new application instance, or the occurrence of certain events. For example, when fire is detected, all relevant buzzers (possibly spread over the entire building) should be activated to alert those present to evacuate the premises. If the command cannot be pushed immediately, it must be ensured that the command reaches its destination when the gateway becomes reachable again.

The Online Service must detect gateway failure and notify the system administrators and the infrastructure owner when such problems persist.

3.2.2 Data synchronization and storage

Data from sensors and actuators needs to be stored on both the gateway and the Online Service. Gateways are resource-constrained and can only store limited historical data from the connected devices. In the Online Service, historical information sent by each sensor and actuator is collected. This historical data allows application instances to, for example, learn the behavior of people working in a building or give a long-term historical overview of monitored conditions. Finally, this data presents interesting opportunities with respect to big data analysis for the SIoTIP Corporation as well as interested third parties.

In this model, sensors and actuators connect, via one or more motes, to a gateway, and all gateways connect to the Online Service to synchronize data. Most synchronization activities are initiated by the gateway, e.g. to push sensor data. However, the Online Service must also be able to push actuation commands and other messages to applications running on the gateways.

3.2.3 Device sharing

As mentioned earlier, the ability to share an IoT infrastructure among multiple customer organizations and applications is one of the main selling points of the SIoTIP offering. Consequently, application instances can manipulate the same infrastructure elements in different, possibly conflicting, ways. For example, one application instance could control door locks based on access cards while another one may decide to automatically open these doors in case of fire in order to evacuate the building smoothly. Similarly, a passive infrared presence sensor can be used by a light management application to turn on the lights in a room when someone enters, but also by a burglary detection application to detect or prevent break-in attempts at night.

SIoTIP should facilitate such sharing, while ensuring that applications do not hinder each other, or misbehave (e.g. accessing devices or sensor data for which the application is not granted rights).

3.2.4 Notifications and alarms

In case of events that are interesting to customer organizations or specific end-users, applications can trigger notifications and alarms. For example, a burglary detection application could trigger an alarm when motion is detected at night. These can be sent to the recipient to a mobile app, e-mail, SMS, or via the customer organization's dashboard depending on the capabilities of the application at hand. The customer organization is able to configure the method of delivery. Furthermore, recipients can request an overview of previously-received notifications and alarms.

Failures in the infrastructure, e.g. a failing sensor, should be reported to the infrastructure owner. System components can trigger notifications for system administrators to inform them of system events that require their attention, such as a misbehaving application (instance) or failing system components.

As alarms indicate critical events that may need immediate attention, their delivery should be as fast as possible.

3.2.5 Access control

Since the infrastructure is shared among multiple users, it is important to authenticate them, verify whether they are authorized to perform the requested actions and prevent users from (actively) influencing application instances from others. More specifically, application providers, infrastructure owners, customer organizations and system administrators need to be authenticated. The system also needs to ensure that customer organizations are only given access to the sensors and actuators that are exposed to them.

3.2.6 Services offered to stakeholders

This section describes the services offered by SIoTIP to the main stakeholders, which is done through interfaces of the Online Service.

Customer organization dashboard Customer organizations interact with the system primarily via the SIoTIP Online Service's customer organization dashboard accessible via a web portal. This dashboard allows customer organizations to (un)subscribe to applications, configure application instances, and other tasks such as consulting their invoices and notifications. Furthermore, customer organizations can manage the roles of the end-users of their application. For example, they can assign certain employees the role of evacuation manager or allow them to control the smart heating via a mobile app.

Infrastructure owner dashboard Infrastructure owners can use this dashboard to monitor and configure their installed hardware and customer organizations. For instance, they can assign (parts of) the installed gateways, sensors and actuators to customer organizations, add new customer organizations or remove an existing one. It also provides access to a hardware store, allowing infrastructure owners to order gateways, motes, sensors, and actuators. Furthermore, infrastructure owners can configure their topology, essential for many applications, via this dashboard. Finally, the infrastructure owner can consult its received notifications and alarms, for example when a sensor or actuator has failed.

System administrator dashboard As SIoTIP system administrators must monitor and manage the entire platform, their dashboard should provide them with the necessary tools. This includes showing information (e.g. resource usage statistics) about running applications, offering the ability to shut applications (or instances) down, receiving and viewing notifications about SIoTIP system components as well as external component failures (e.g. failed gateways).

Application provider dashboard Finally, application providers have access to their own dashboard. It enables them to get an overview of their applications and provides the ability to upload new and update existing applications. As the uploaded applications are tested online/automatically, the dashboard also presents to them the status of this application check and points out potential issues that may impede deployment.

Service Level Agreements and legal constraints The SIoTIP organization offers a Service Level Agreement (SLA) enclosed as a part of the provided support towards our customer organizations. This agreement stipulates certain quality requirements (in terms of performance, availability, security and privacy).

A number of contracts will have to be negotiated with other stakeholders. Some of these will be with application providers, stating some constraints their applications have to conform to. In return we provide certain availability and reliability guarantees to application providers. This includes, among others, that the SIoTIP Online Service must be almost permanently available and that sensor data is never lost or inconsistent. Similarly, a contract with the telecom operator should ensure the desired degree of internet connectivity.

Some legal constraints should also be taken into account. The system will be used within corporations and will collect, process and store business data. Hence, the way in which SIoTIP deals with this data must be in line with security and privacy regulations such as GDPR.

3.3 SIoTIP Applications

The primary goal of SIoTIP is to support the development and deployment of applications which leverage the IoT infrastructure made accessible through the platform. For customer organizations, the application they subscribe to are the primary way to interact with the platform, and thus also to leverage the installed sensors and actuators.

3.3.1 Distributed applications

As described, these applications are developed and maintained by third-party application providers and often domain-specific.

SIoTIP applications are inherently *distributed*. For instance, a smart heating application likely consists of several parts cooperating with each other in some way. The local parts running on gateways can monitor and, if needed, adjust the temperature in specific room by interacting with the connected sensors and actuators. The algorithms used to learn the behavior and daily routine of the people in a building typically requires large amounts of historical data, and are thus executed on the Online Service. This Online Service part thus uses the sensor data provided by the application parts running on the gateways and inform them of necessary changes, e.g. start heating a specific room earlier to accommodate early birds. Finally, some persons, e.g. the employees of the customer organization, can be allowed to manually adjust the temperature via a mobile app, i.e. a part of the application running on their smartphone.

The main added value of SIoTIP is that it supports the application providers in developing and deploying such complex, distributed applications by taking care of the distribution and communication between different parts of an application. Therefore, SIoTIP must provide services or APIs to accommodate such a distributed applications, and deal with all encompassing communication between different parts of a running application instance.

3.3.2 Device access

Application providers expect a reliable platform to deploy their applications on. A single broken sensor, for example, should not necessarily stop an application instance entirely or even hamper its correct functioning. SIoTIP has to provide APIs both for code running on the gateway and on the Online Service.

3.3.3 Application life-cycle

As software continuously evolves, application providers should be able to update their applications to newer versions including bug fixes and/or new features. They have the freedom to choose for updates whether all existing instances of the applications are automatically updated to the new version (e.g. a minor bug fix update) or to require customer organizations to explicitly update to the new version (e.g. a major feature update which requires a new subscription). In both cases, SIoTIP should automatically update all parts of the affected application instances seamlessly. Note that in the latter case, instances of different versions of the same application can co-exist.

For application providers, a development, testing, and debugging environment is provided. This environment provides easy access to all APIs provided by SIoTIP. Furthermore, it allows to emulate a more realistic deployment environment, e.g. mimicking incoming sensor data, to test and debug the application under development.

The system has to check new or updated applications to ensure that they, for example, cannot cause a system crash, contain memory leaks, or misbehave in any other way. Therefore, when uploaded, each application is automatically tested in a sandbox environment. If all tests are successful, the application is automatically made available, and otherwise a SIoTIP system administrator performs a secondary review and decides whether to accept or reject the application. In the latter case, the application provider is also notified of the reason for rejection. As mentioned before, application providers are given the choice between automatically updating existing application instances or requiring customer organizations to subscribe to them explicitly.

The system should monitor the execution of application instances. This yields information regarding the performance of applications, such as the amount of CPU and memory used by each application. This can be complemented with application-specific information, such as when an application is executing a learning algorithm and hence can be expected to utilize more resources. In case of problems (e.g. an application instance misbehaving due to a bug), the system must notify a system administrator, who can (temporarily) shut down affected applications instances. Furtermore, the corresponding application provider should be notified of possible problems with his or her application.

3.3.4 Sample applications

This section describes some sample applications for building control. It is outside the scope of this document to provide a detailed, algorithmic description of each application and these are meant as illustrative examples of how SIoTIP can be leveraged.

Fire control application The fire control application uses sensors to detect fire and takes suitable action, e.g. sounding buzzers as warning signal and issuing notifications.

Fire can be detected by (a combination of) air temperature and humidity sensors, surface temperature sensors, and O_2 gas sensors. For an application instance to be activated, each room should be equipped with at least two of these sensors. Furthermore, fire alarm buttons can also be integrated using, for example, power socket actuators. In case a fire is detected, a first task of the system is to alert anyone present by, for example, activating the available buzzers or activating alarm bells using power socket actuators. Based on the topology, doors nearby the affected area can be unlocked to simplify evacuation and any available sprinklers can be activated. Note that when unlocking doors, the fire control application possibly overrides any active access control system. Furthermore, the local fire department or emergency response team can be automatically notified.

Building access control application This application allows to control access to rooms and areas of a building by controlling door locks, e.g. if some areas are only accessible for authorized personnel. Thus access can be granted or refused after someone authenticates him- or herself using, for example, a badge reader or by entering a code on a key pad. Therefore, such a (custom) sensor must be installed near each controlled door and each such door has to be equipped with a mechanism to lock and unlock it, e.g. an electromagnet that can be turned on and off via a power socket actuator.

Furthermore, different end-users can be involved. For example, a security guard must check whether each door on his or her route is indeed locked and intermediately sends status reports via a mobile app. Furthermore, it is possible to differentiate the actions performed by the application based on who authenticates and/or where they authenticate. For example, a notification can be sent to the security guard when an unauthorized person tries to enter a specific room. Depending on the authentication method used and data stored, this application can also provide a log of who entered specific areas and when.

Burglary detection application The burglary detection application detects break-in attempts in the monitored area. End-users could arm and disarm the system via a mobile app or via an attached IoT device such as a console near an entrance and exit.

Burglary attempts can be detected by monitoring presence and by sensors installed on doors and windows. Thus at least one sensor that can detect presence of a person should be installed in each monitored room. Furthermore, all windows and doors providing access to a monitored area should be equipped with accelerometers to detect when they are opened.

Upon detecting a break-in, a notification is sent to the relevant parties, e.g. the security guard(s) on duty or the local police department. Furthermore, actuators can be triggered to activate an alarm and/or cameras.

Smart heating application The heating application allows customer organizations to control the heating (and cooling) system in (a part of) the building in a (semi-)automated way. Customer organizations can specify general policies about, for example, the minimum and maximum temperature. The heating application will control the heating system based on received temperature sensor readings, detected motion of people, weather predictions and the policy of the customer organization for that area of the building. Therefore, sufficient sensors should be available in each room to at least sense the temperature and presence of people.

Scenarios such as the following are supported:

- The temperature within a room can be kept within an interval defined by the customer organization.
- In empty rooms, a minimum temperature is maintained at all times, but the temperature is increased when motion is detected in order to accommodate the comfort of persons in the room.
- The application could include learning behavior, learning about the daily routine of employees in an office. This allows the system to pre-heat the room prior to the employees their arrival.
- End-users can manually adjust, within the bounds of the enforced policy, the settings via, for example, a mobile app or a web portal.

Light management application The light management application provides control of the lighting conditions in areas of the building. This is based on the time of the day, daylight levels, presence of people and the preferences of the customer organization for that area of the building. Furthermore, based on the available topology, lights can be switched on before the room itself is reached, e.g. when detecting motion in an adjacent hallway.

Scenarios such as the following are supported:

- If the light sensor detects that it is getting dark outside and people are present, the lights are switched on.
- The customer organization can modify the daylight illumination level sensitivity via their online dashboard.
- The customer organization can indicate that certain light bulbs (e.g. toilet lights, corridor lights) should only be switched on when motion is detected.

All lights managed by this application have to be equipped with a power socket actuator. Furthermore, each relevant room should be equipped with sensors for motion detection or there should be such sensors nearby, e.g. in the adjacent hallway.

A Appendix: Supported MicroPnP devices

This appendix provides an overview of the sensors and actuators currently supported by VersaSense as well as the accompanying MicroPnP support material. Table 1 describes the capabilities of the supported sensors and actuators as well as how they can be configured. Table 2 lists additional MicroPnP materials which can be used to construct richer infrastructures.

Table 1: Summary of currently available MicroPnP sensors and actuators.

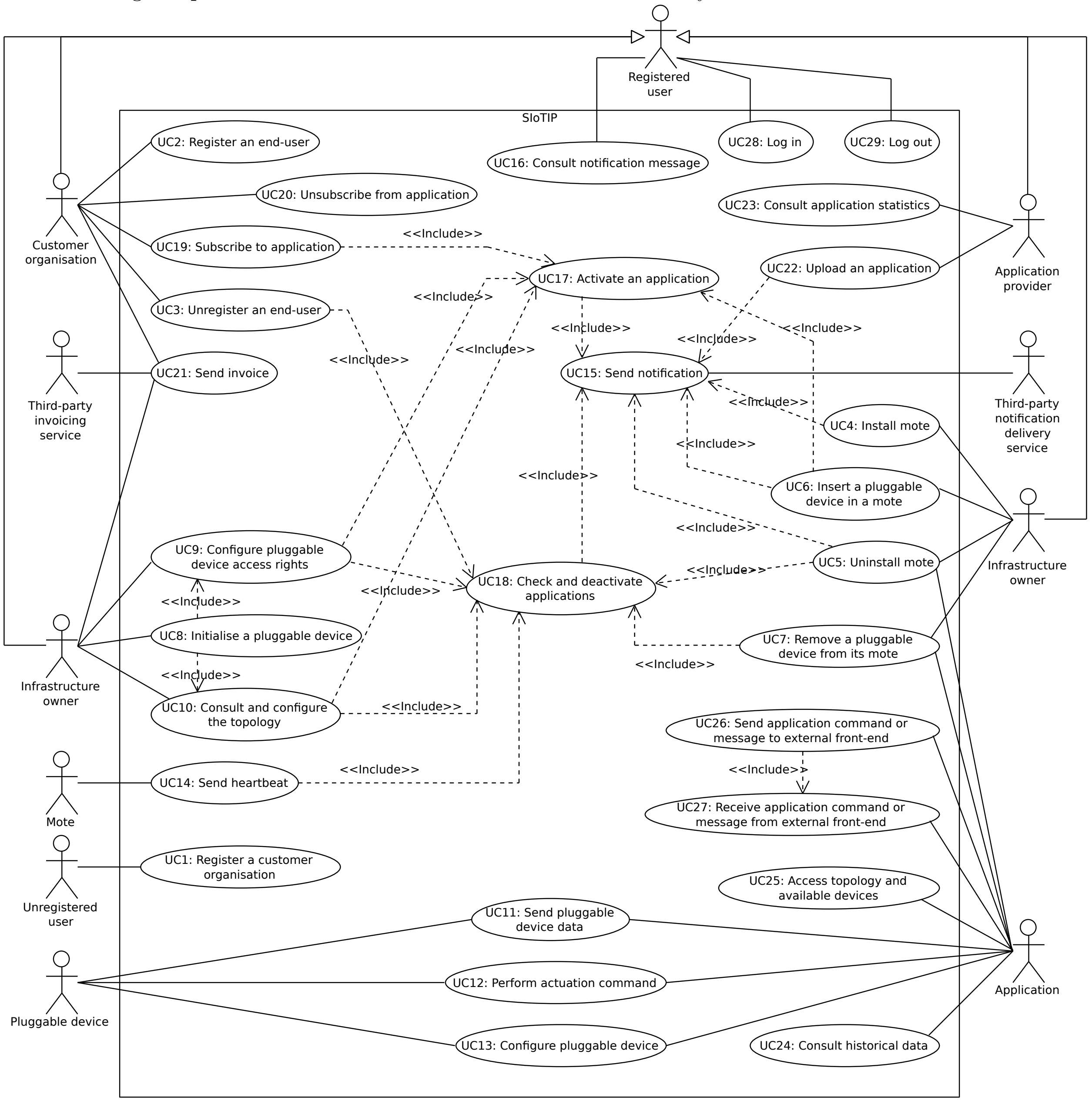
Name	**Description and important details**	**Configuration**
Passive Infrared (Presence) Sensor	Allows detection of motion, and hence presence of people. This is done by regularly taking a low-resolution infrared picture and comparing it to the previous picture taken. The sensor sends a “1” upon detecting motion.	None
Accelerometer	Measures acceleration in 3 dimensions as a tuple of floating point numbers: “(*x-axis,y-axis,z-axis*)”. Each number represents the acceleration component in that dimension, is measured in m/s^2. All-weather proof.	Measurement interval (default 0.5s)
Air Temperature and Humidity Sensor	Measures air temperature and air humidity. The temperature is expressed in °C (range -40 up to 125°C with a precision of 0.2°C). Air humidity is expressed as relative humidity in % (precision 0.1, a comfort level is usually between 45 and 55%).	Measurement interval (default 10s)
Surface Temperature Sensor	Measures the temperature of the surface it is attached to. The temperature is expressed in °C (range -73 up to 260°C with a precision of 1.5°C)	Measurement interval (default 10s)
Microphone Sensor	Allows detection of sound, and hence presence of people. Returns a “0”” or “1”” depending on whether or not the amount of detected noise/sound is above a given threshold (in dB). Detection range is between 20 dB and 120 dB with a precision of 0.1 dB.	Measurement interval (default 5s), noise threshold (default 40 dB).
Air Pressure Sensor	Measures air pressure in bar. Range is between 0.10 and 1.15 bar. The precision is 0.00001 bar. (A normal level is around 1.01325 bar)	Measurement interval (default 10s)
Light Sensor	Measures the amount of light in LUX. Range 20 up to 20000 LUX. The precision is 1 LUX (A normally lit room is usually between 300 and 600 LUX).	Measurement interval. (default 10s)
Distance Sensor	Measures a distance in cm. The sensor detects distances between 2 and 350 cm with a precision of 0.1cm. The measurement is done by sending an infrared light beam and measuring the time it takes for the light beam to be reflected back to the sensor.	Measurement interval (default 5s)
O_2 Gas Sensor	Measures the amount of O_2 gas in the air, in % with a precision of 0.01. All-weather proof. Requires external power supply.	Measurement interval (default 5s)
Buzzer Actuator	Sounds a buzzer. Put a “0” or “1” for turning off or on, respectively.	None
Power Socket Actuator	Turns a power socket on/off (by putting “0”/“1”), hence controlling the power supply of some other device.	None

Table 2: Summary of MicroPnP support material.

Name	Description and important details
SmartMesh IP Indoor Board (the *MicroPnP mote*)	A board with peripherals for connecting up to 3 sensors/actuators.
MicroPnP Network Manager	Network manager allowing connectivity with up to 100 motes. The interconnection between the gateway and the motes.
SmartMesh IP Range Extender	Range extender for indoor or outdoor use.
Weather-proof armor	Durable, weather-proof armor for protecting motes.

B Appendix: Use case diagram

The use case diagram presented below summarizes the main functionality of SIoTIP.


Registered user
SIoTIP
UC2: Register an end-user
UC16: Consult notification message
UC28: Log in
UC29: Log out
UC20: Unsubscribe from application
UC23: Consult application statistics
UC19: Subscribe to application
UC17: Activate an application
UC22: Upload an application
UC3: Unregister an end-user
UC15: Send notification
UC21: Send invoice
UC4: Install mote
UC6: Insert a pluggable device in a mote
UC9: Configure pluggable device access rights
UC18: Check and deactivate applications
UC5: Uninstall mote
UC8: Initialise a pluggable device
UC7: Remove a pluggable device from its mote
UC10: Consult and configure the topology
UC26: Send application command or message to external front-end
UC14: Send heartbeat
UC27: Receive application command or message from external front-end
UC1: Register a customer organisation
UC25: Access topology and available devices
UC11: Send pluggable device data
UC12: Perform actuation command
UC13: Configure pluggable device
UC24: Consult historical data
<<Include>>
Customer organisation
Third-party invoicing service
Infrastructure owner
Mote
Unregistered user
Pluggable device
Application provider
Third-party notification delivery service
Infrastructure owner
Application

C Appendix: MicroPnP interfaces

Note: *The operations in these interfaces do not raise any exceptions, so sending incorrect commands has no effect.*

C.1 Gateway – Provided interfaces

- Heartbeat
  - `void heartbeat(MoteInfo moteInfo, List<Tuple<PluggableDeviceID, PluggableDeviceType>> pds)`
    - Effect: Periodic heartbeat from the mote to the gateway, including a list of the `Pluggable-Devices` and their `DeviceTypes` (i.e. those currently plugged into the mote)
    - Exceptions: None
- SensorData
  - `void rcvData(PluggableDeviceID pID, SensorReading sensorReading)`
    - Effect: Provide sensor data to the gateway *(Periodic)*.
    - Exceptions: None
  - `void rcvData(PluggableDeviceID pID, SensorReading sensorReading, int requestID)`
    - Effect: Provide requested sensor data to the gateway *(Callback)*.
    - Exceptions: None
  - `void rcvActuationCallback(int requestID)`
    - Effect: Confirm execution of an actuation command.
    - Exceptions: None
- DeviceManagement
  - `void pluggableDevicePluggedIn(MoteInfo mInfo, PluggableDeviceID pID)`
    - Effect: Notify the gateway that a new `PluggableDevice` is connected.
    - Exceptions: None
  - `void pluggableDeviceRemoved(PluggableDeviceID pID)`
    - Effect: Notify the gateway that a `PluggableDevice` is removed.
    - Exceptions: None

C.2 Sensor – Provided interfaces

- Config
  - `Map<String,String> getConfig()`
    - Effect: Returns the current configuration of a `PluggableDevice` as a map.
    - Exceptions: None.
  - `boolean setConfig(Map<String,String> config)`
    - Effect: Changes to configuration of the `PluggableDevice`. Sending values to an incorrect `PluggableDevice` (e.g., 'resolution','640x480' to the buzzer) has no effect.
    - Exceptions: None.
- RequestData
  - `SensorReading getData()`
    - Effect: Synchronously retrieve the current value of the sensor.
    - Exceptions: None.
  - `boolean getData(int requestID)`
    - Effect: Asynchronously retrieve the current value of the sensor *(Callback)*.
    - Exceptions: None.

C.3 Actuator – Provided interfaces

- Config
  - `Map<String,String> getConfig()`
    - Effect: Returns the current configuration of a `PluggableDevice` as a map.
    - Exceptions: None.
  - `boolean setConfig(Map<String,String> config)`
    - Effect: Changes to configuration of the `PluggableDevice`. Sending values to an incorrect `PluggableDevice` (e.g., 'resolution','640x480' to the buzzer) has no effect.
    - Exceptions: None.
- Actuate
  - `boolean sendActuationCommand(String commandName)`
    - Effect: Send an actuation command to the actuator. The actuator acknowledges execution of the command. Sending an incorrect actuation command (e.g. 'take picture' to a 'buzzer') has no effect.
    - Exceptions: None.
  - `boolean sendAsyncActuationCommand(String commandName, int requestID)`
    - Effect: Send an actuation command to the actuator. *(Callback)* The actuator acknowledges receipt of the command. Sending an incorrect actuation command (e.g. 'take picture' to a 'buzzer') has no effect.
    - Exceptions: None.

C.4 Datatypes

- `MoteInfo`: A object containing information on a mote. This is a list of key-value pairs. The values depend on the type of mote. For example, only a battery-powered mote would include the batterylevel info.
  - `int moteID`
  - `int manufacturerID`
  - `int productID`
  - `int batteryLevel` *(Only for battery-powered motes.)*
- `PluggableDeviceID`: A unique identifier of the `PluggableDevice`.
- `PluggableDeviceType`: Denotes the type of the pluggable device.
- `SensorReading`: Sensor values. The data is encapsulated within a JSON message, and should be converted by the gateway into something meaningful based on the type of the originator (contained in the message) of the data.
- `Map<String,String>`: A map of key-value pairs. E.g., 'sensitivity','10' or 'resolution', '640x480'.