The Space Broker: A Middleware for Mediating Interactions in Smart IoT Spaces

Hamim Md Adal
Department of Computer Science
University of New Mexico, USA
hmdadal@unm.edu

Colin Milhaupt
Department of Computer Science
University of New Mexico, USA
cmilhaupt@unm.edu

Jie Hua
Department of Electrical and Computer Engineering,
University of Texas at Austin, USA
mich94hj@utexas.edu

Christine Julien
Department of Electrical and Computer Engineering
University of Texas at Austin, USA
c.julien@utexas.edu

Gruia-Catalin Roman
Department of Computer Science
University of New Mexico, USA
gcroman@unm.edu

ABSTRACT

The Internet of Things (IoT) is a major technological development likely to have a profound effect on all aspects of society. Among other things, it promises smooth and personalized interactions between people and the spaces they inhabit and visit. Unfortunately, we are not yet at the point of interacting with smart spaces *per se*; rather, we simply interact with collections of devices having different interfaces, offered by different manufacturers, living in different administrative domains, and using different apps. The research reported in this paper promises to take us a step closer to achieving the personalized interaction modalities the IoT technology is capable of offering. The starting point is to reimagine the smart space as being defined by spatial characteristics (e.g., illumination, security, temperature, etc.) with most devices receding from the user's explicit awareness. Users can specify their needs from the environment in terms of the abstract characteristics. Key to accomplishing this is the introduction of the concept of the *Space Broker*, a software agent that manages available devices so as to meet user requirements expressed in terms of spatial characteristics.

CCS CONCEPTS

• **Human-centered computing** → **Mobile phones**; **Mobile devices**; **Contextual design**; **Collaborative interaction**; • **Information systems** → **Location based services**; **Sensor networks**; • **Computing methodologies** → **Intelligent agents**.

KEYWORDS

space, characteristic, middleware, space broker, smart, api, device, sensor, application, agent, interaction, IoT, android

BuildSys '21, November 17–18, 2021, Coimbra, Portugal

ACM ISBN 978-1-4503-9114-6/21/11…$15.00
https://doi.org/10.1145/3486611.3486664

ACM Reference Format:
Hamim Md Adal, Colin Milhaupt, Jie Hua, Christine Julien, and Gruia-Catalin Roman. 2021. The Space Broker: A Middleware for Mediating Interactions in Smart IoT Spaces. In *ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys '21), November 17–18, 2021, Coimbra, Portugal.* , 10 pages. https://doi.org/10.1145/3486611.3486664

1 INTRODUCTION

In a smart space, personal devices and IoT-enabled physical devices interact intelligently in a connected, organized, and programmable digital environment. The objective is to make the user interaction natural and intuitive. The Internet of Things (IoT) provides such a platform where numerous programmable devices and sensors offer services designed to meet specific users' needs in the context of the spaces they inhabit or explore. In a smart IoT environment, personal devices (such as smart mobile phones, smart tablets, etc.) and physical devices (such as smart locks, smart lights, smart speakers, etc.) work together by communicating over a local network to achieve complex functionalities centered on the user's needs.

Users' personal devices are no longer restricted merely to making calls and sending texts but can also energize a wide range of smart features in IoT spaces. A user can manually set up a lighting device via a mobile app and control it over a network. In the same way, a user can set up and use a smart speaker. Despite the availability of such a plethora of features, providing them with minimal human involvement and cognitive load remains a challenge.

To take advantage of today's IoT environments, a user needs to know the whereabouts of every IoT-enabled physical device in the space, the specific services each device offers, and the details of how to interact with the device. Manually installing each physical device is yet another struggle. It is difficult to maintain a single interface when a user moves from one space to another, especially in spaces that are recognized through physical devices. When users visit new spaces, they discover new sets of devices with different configurations and capabilities. Existing frameworks providing proprietary interfaces between personal devices and physical devices do not transfer readily from one environment to another, even when the same manufacturer is involved. After inspecting all the complexities, an ideal solution demands having one standard interface that

creates a cohesive view of the physical space in which multiple devices of different types or from different manufacturers coordinate to ensure a personalized and carefree experience for the user.

A user should perceive the IoT environment as a smart space that offers services that impact *spatial characteristics* (e.g., illumination, temperature, sound) of the environment rather than as a set of physical devices that affect the environment. Such a transition requires an agent that acts as an intermediary between the user and the space, accepting requests from the user in terms of spatial characteristics and realizing these requests using the (potentially dynamic) set of available physical devices. We propose the *Space Broker* as an agent that provides this mediation between high-level, abstract user requests (in terms of spatial characteristics) and the set of devices and sensors available at a particular location. For instance, the need for an ambient temperature level at the kitchen may be addressed by engaging a combination of devices quite distinct in nature (e.g., smart HVAC, smart window shades) as well as the impact of unrelated devices (e.g., heat produced by a stove that can affect the kitchen's temperature) that happen to be in use at the time. Users no longer need to maintain knowledge about the space and the physical devices it contains. All they need to do is to make a request in terms of spatial characteristics, and the *Space Broker* will adjust the devices using its knowledge about the space.

To illustrate the role of the Space Broker, consider a user Alice whose home has programmable light devices, a digitally controllable HVAC system, and a variety of sensors in various locations in her home. When she arrives home, a handshake takes place between her personal device and the Space Broker. As a result, the personal device provides the Space Broker access to its available onboard devices (e.g., camera, light sensor). In addition, the Space Broker offers Alice full access to all of the spatial characteristics it controls.

Now suppose Alice needs a specific amount of illumination to read a book while sitting on her chair in her living room. She also appreciates an ambient temperature of $75^{\circ}F$. The living room's thermostat provides access to the temperature, but it does not have a light sensor to measure ambient luminance. The Space Broker utilizes the thermostat to control the temperature but still needs to find a solution for the illumination. Luckily, Alice's personal device has a camera that can measure luminance in the space. Therefore, the Space Broker employs Alice's personal device's camera (given that Alice permits the Space Broker to access it) to measure the illumination level in the room and then uses this information to operate the programmable lighting devices to achieve the intended amount of luminosity. The Space Broker meets Alice's requests by utilizing its knowledge of and control over the physical devices in the space. One important observation is that Alice is not aware of the light devices, HVAC, or sensors in the space; she is only aware of the characteristics of the space made manifest by the Space Broker.

The *Space Broker* provides a single interface to connect indirectly and transparently to all available physical devices and sensors in a space and to assist a user in interacting with a space in terms of characteristics that can be inspected and regulated. In this context, this paper makes the following key research contributions:

- We support personal devices viewing a space not as a collection of IoT devices to be managed, but in terms of a set of abstract spatial characteristics that can be examined and controlled.
- We propose a software agent, the *Space Broker*, to support this view, whose role is to manage interactions with devices by translating requests about spatial characteristics into actions taken by one or more devices in the space.
- We design the *Space Broker* API, which allows user-facing applications to interact with the space at the level of characteristics. The interaction process is simple and managed on the user-device side by a lightweight proxy that makes developing applications more effortless and less communication intensive.
- We demonstrate the feasibility of our approach within a small-scale physical environment.

The next section presents related work. In Section 3, we describe the *Space Broker* model, including a discussion of *spatial characteristics* and their relationships to physical devices. Section 4 presents the Space Broker API. In Section 5, we define a small-scale prototype IoT environment. In Section 6, we compare our *Space Broker* model with existing approaches. Section 7 provides a discussion of future work, and Section 8 concludes.

2 RELATED WORK

Relying on a middleware to mediate interactions between a user and IoT devices is not a new idea. The Semantic Information Broker (SIB) [8, 11, 12, 16] adopts the semantic web to share information between devices and enable portability and dependability. Although SIB simplifies development, it does not separate a user's preferences from the characteristics of the space and thus an interface employed to interact with one space cannot efficiently transfer to other spaces (because different spaces present different semantic models).

Information Centric Networking (ICN) has also been used to cope with IoT challenges. ICN was originally used to manage data collected by IoT devices but has also recently been investigated to provide solutions to IoT device organization [1–4, 17]. In ICN, all IoT devices are treated as resources and referenced by names; this approach increases programming simplicity and interoperability. The user does not need to form a direct connection to a device to use it. However, the user still needs to provide control signals at the individual device level by specifying the exact name of the resource to use so that the ICN infrastructure can manage the interaction. A similar but more abstract idea is the Service Centric Network (SCN) [5], which enhances ICN by supporting services as an extension to devices so that a user does not have to reference specific devices. However SCN only supports a predefined set of services and the number of services grows exponentially relative to the number of devices, so the approach does not easily scale.

We target more ubiquitous interactions in which a user is unaware of the underlying device that satisfies a request. The *Space Broker* abstraction enables this separation by relying on existing solutions to provide connectivity among heterogeneous devices. UniGate [7] is a universal gateway that can be implemented on the wireless router that is common in an IoT-enabled space like those we target. Other middleware like Hydra [6] and Warble [15] can provide automatic discovery of devices and dynamic connection based on the current IoT context. Such approaches are more suitable for our needs. By layering on top of such middleware, the Space Broker can dynamically leverage the sensors and actuators

available in the space, even those that are transient, e.g., because they reside on the personal devices that users bring into the space.

Since the client of the Space Broker is the personal device which is a proxy for a specific user, the smart space is expected to be responsive to the presence of the user. That is, the Space Broker model should be able to support responsive services. Examples of such services are introduced in many smart space applications and systems. rIoT [10], CA4IOT [14] and ACE [13] use sensor collected context to enable automation in the IoT. In our model, such a system can be deployed on the personal device to query the Space Broker for sensor readings and issue requests to the Space Broker in response to the user's presence or immediate needs.

3 SPACE BROKER MODEL

In this section, we start by presenting a conceptual overview of the Space Broker model. We then provide a formal definition of the model that leans on the definition of *spatial characteristics* to represent the context of the smart space.

Conceptual View. Interactions between personal devices and IoT-enabeld physical devices are often supported by a local network accessible through a router or wireless gateway. The IoT vision promises an immersive environment that support users' needs in a spontaneous way. But today, users manage devices in isolation and interact with them in proprietary ways. This is because users directly control individual devices, rather than controlling or influencing *the space* in which they exist. Users also have to deal with difficulties caused by proprietary interfaces promoted by different manufacturers. To overcome these challenges, our Space Broker promotes a model in which interactions happen at a different cognitive level where users interact with the space and not with specific devices. For example, when reading a book, the user may require a certain level of illumination and ambient temperature at a particular location. Fulfilling this request demands a software agent that can (1) identify multiple devices related to a particular functionality (i.e., categorizing light-influencing devices and temperature-influencing devices separately to achieve to the tasks of providing illumination and temperature, respectively) and (2) control those devices in specific ways to provide the requested service. This is a dramatic shift in the way users are able to interact with their smart spaces: rather than a user activating "the light device above the counter" this shift allows the user to focus on the more abstract goal of "illuminating the kitchen for the purpose of cooking."

Figure 1 shows the conceptual model of the Space Broker. An instance of the Space Broker is associated with a bounded space (e.g., a home, an office, a city block, etc.) and provides a virtual representation of that space in terms of concepts we call *spatial characteristics* (e.g., illumination, temperature, security). We contend that most user concerns can be addressed at this level rather than at the level of specific individual devices. In our Space Broker model, we rely on the user's *personal device* (e.g. smartphone) to provide the interaction interface with the characteristics of the surrounding space. Through the personal device, the user can query for the status of available spatial characteristics and make requests for changes to those same characteristics. The characteristics can be queried, modified, and maintained upon request in response to the user needs and at specific locations.

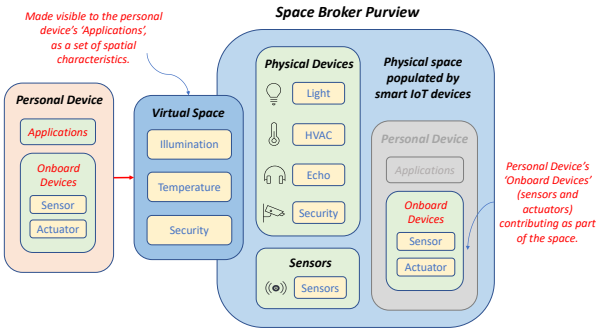


Figure 1: Conceptual Model of IoT Space Broker.

To satisfy users' queries and requests, the Space Broker employs the IoT-enabled physical devices that reside in the space. The Space Broker has knowledge of and access to all devices (regardless of the manufacturers) in the space, including the personal device's onboard devices (such as sensors and actuators, which may be made available to the Space Broker depending on the user's permission). To participate in the space, each physical device needs to only discover the local Space Broker and announce itself. The Space Broker maintains knowledge of the details (such as location and functionality) of every physical device in the space and integrates their capabilities into the space. Users do not need to be aware of any particular physical devices in the space, they only have to concern themselves with the characteristics of the space. Of course, if the Space Broker is leveraging the personal device's onboard devices to support some characteristic, the user may need to be aware of that purpose, for instance to ensure that the device is not in their pocket while being used to sense the illumination level.

When a personal device makes requests for a specific characteristic, the Space Broker acts on them by reading and controlling the appropriate set of devices, given the configuration of the space and the current contextual state of the space. In this paper, though we focus on the abstract interactions with the space in terms of characteristics, we acknowledge that there are situations when the user will want to interact with specific devices. Our view of the Space Broker is not incompatible with a system that ultimately presents a user with both the device-level and characteristic-level concerns. Because the device-specific coordination is widely explored and well-handled, in this paper, we have our singular focus on users interacting with the space at the level of characteristics only.

Formal Specification. We next offer a formal specification for the functionality of the Space Broker. We assume that the capability of a device to sense and change a characteristic does not change over time. While the types of spatial characteristics one can envision are many and varied, a reasonable starting point is to view the Space Broker *SB* as a set of functions mapping space to scalar ranges:

$$SB = \{F_1, \ldots, F_n\}, \text{where } F_i : S \rightarrow R_i^+$$

where each element of S is a location in the space, and each function F_i (associated with the i^{th} characteristic) maps point in space to a value in the scalar range R_i plus undefined ($\perp$).

Despite its immediate appeal, this formalization is overly simplistic because it ignores the basic fact that the definitions of the functions F_i are affected by the state of the devices residing in the space. If we assume that there are m devices (d_j with $j = 1 \ldots m$) in the space and each device has a state determined by a control variable v_j, the space broker SB is actually a set of functions that map control variable values (a control configuration) to functions from space to scalar ranges:

$$SB = \{FF_1, \ldots, FF_n\} \text{ where } FF_i : [V_1 \times \cdots \times V_m] \rightarrow [S \rightarrow R_i^+]$$

with V_j being the domain of the control variable v_j. This revised definition captures the fact that the control variables impact the mapping of spatial characteristics to scalar values. Consequently, when a personal device issues a query for the third characteristic of the space at location x, the query might be represented as $F_3(x)$, but this is internally converted by the Space Broker to $FF_3[v_1, \ldots, v_m](x)$. Similarly, for a request to change the state of the space to achieve a scalar value r for this characteristic at location x, the user's request would be phrased as $F_3(x/r)$, which is converted by the Space Broker as a request to solve the equation:

$$FF_3[v_1, \ldots, v_m](x) = r \text{ for control variables } v_1, \ldots, v_m$$

leaving unchanged all the control variables that cannot affect the solution.

If, for instance, F_3 refers to illumination and the only devices affecting illumination are d_1 and d_3, all the other control variables could be denoted as "don't care" arguments as in:

$$FF_3[v_1, -, v_3, -, \ldots, -](x) = r \tag{1}$$

Solving these kinds of equations is at the core of implementing the Space Broker. However, the efficiency and accuracy of the associated processes is also important. For now we perform an exhaustive search of available solutions, which is tractable for small smart home deployments. As the scale of purview of the Space Broker grows, heuristics will also be needed to search for good (even if not optimal) solutions efficiently. The development of effective algorithms that solve these equations is essential to deploying the Space Broker for large scale smart spaces, which is considered as future work. In principle, these approaches may involve exploiting sensors in the space, learning from previous experiences, exploiting high accuracy physics models, selecting promising starting points for localized searches, constraint optimization, etc. In this paper, rather than focusing on optimized algorithms, we concern ourselves with the programming interfaces the Space Broker presents to applications wishing to control a small-scale smart space.

4 PRAGMATICS OF CAPTURING CHARACTERISTICS

The user's personal device sees the space as a set of spatial characteristics offered by the Space Broker, whereas the Space Broker views the space as a collection of physical devices and sensors responsible for delivering services. As far as the user requests are concerned, the Space Broker supports two kinds of functionalities. First, it responds to queries made by the user regarding the current values of particular spatial characteristics. Second, it acts upon user requests for changes in the value of characteristics (to modify a characteristic's value or to maintain a given value over time). Both queries and requests relate to specific locations in the space.

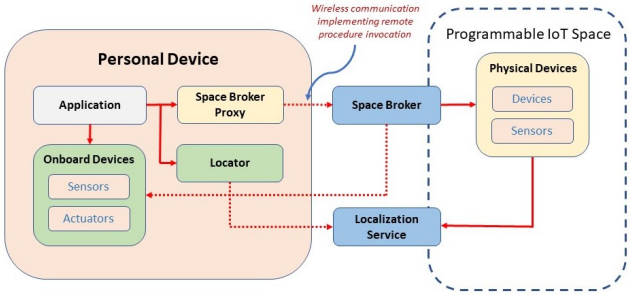


Figure 2: Architecture of the IoT Space Broker Model.

Location is a fundamental concept in the Space Broker, and the ability for the Space Broker to localize devices (both physical devices in the IoT space and personal devices of users) is essential. However, different spaces will have different resources available to support localization (e.g., overhead cameras, indoor radio-based tracking systems, or dead-reckoning positioning systems). Therefore the Space Broker does not assume a single approach to localization but rather relies on an external *Localization Service* that (1) provides a coordinate system that applications can use for reference in the given space and (2) provides an interface that maps a device's identity onto that coordinate system. If device positions change over time, we assume that the localization service updates this information so that it can provide the Space Broker a continuous view of devices' locations. These assumptions are realizable using smart space localization services that are available today [9].

Figure 2 captures the high-level software architecture supporting users' interactions with the programmable IoT space. As the figure shows, the Space Broker mediates the users' interactions by controlling all of the physical devices in the space, including knowledge of their locations. As described previously, when a personal device enters the space, a handshake process ensues that allows the user's device and the Space Broker to recognize each other.

A skeleton of the Space Broker interaction methods is shown in Table 1. The first two methods listed in the table are callbacks invoked on the Space Broker by the underlying discovery service whenever a new personal device is discovered in the space (i.e., `spaceBroker.enter`) or when a personal device leaves (i.e., `spaceBroker.exit`). During the handshake process that ensues, the personal device and space broker exchange information about the sensing and actuation resources available on the personal device through the `device.requestResources` method. In addition, the Space Broker delivers a piece of code that we term the *Space Broker Proxy* to the user's personal device (using the `device.deliverProxy` method). Through this proxy, the Space Broker makes the user's applications aware of its available characteristics (see `proxy.getCharacteristics`).

Each Space Broker action is tied to a spatial characteristic. Once the personal device acquires knowledge of a characteristic via the proxy, applications on the device can secure access to three modes of interaction with respect to that characteristic. First, the personal

Table 1: Space Broker API

API Method	Description
Discovery-Centered Methods	
`spaceBroker.enter(device)`	When a device enters the administrative domain of the Space Broker, a handshake process between the two ensues, which includes calling `device.requestResources` and `proxy.getCharacteristics()`.
`spaceBroker.exit(device)`	When a known device leaves the administrative domain of the Space Broker, the Space Broker cleans up its registered resources.
Personal Device Integration Methods	
`device.requestResources()`	When this method is called by the Space Broker, the personal device returns proxies to the interfaces of any sensors and actuators the personal device is willing to share with the space along with their corresponding characteristics.
`device.deliverProxy(proxy)`	This method is called by the Space Broker to share with the personal device the proxy that personal applications will use to interact with the space.
Proxy and Characteristic Interaction Methods	
`proxy.getCharacteristics()`	This method returns a list of characteristics that can be queried, modified or maintained in this space. Each characteristic includes its semantic description as well as the capabilities (i.e., read or write) associated with that characteristic.
`characteristic.query(location)`	This method is called by a personal device and returns the value of the characteristic at the provided location.
`characteristic.modify(location, value)`	This method is called by a personal device and requests a change to the specified characteristic at the given location. The value is the target number for the characteristic to reach.
`characteristic.maintain(location, value)`	This method is called by a personal device to maintain the value of the characteristic at the given location at value. This method can also optionally be passed an expiration time after which the Space Broker stops maintaining the value.

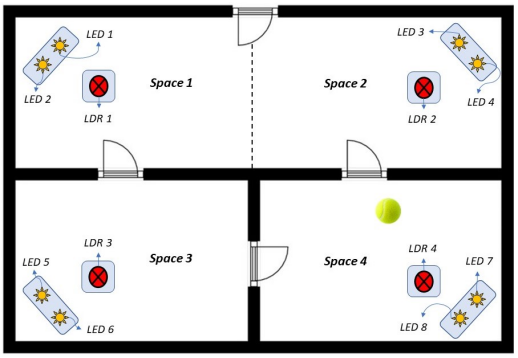
device can query the state of the space relative to that characteristic (see `characteristic.query` in Table 1); the illumination level at a specified location can be accessed in this manner.

Second, if a characteristic is modifiable, the application can request to change the state of the characteristic to a specified value at a particular location (see `characteristic.modify` in Table 1). For instance, a user can request the Space Broker to deliver 850 lumens of luminosity in the kitchen to prepare supper. Comparing the current and desired illumination level, the Space Broker adjusts the available light devices in the kitchen to achieve the intended amount of luminance for the user.

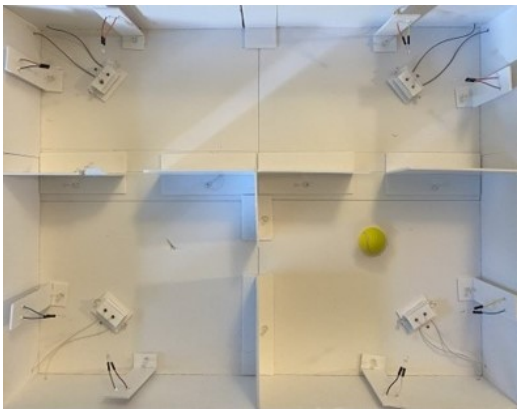
Finally, if the characteristic is maintainable, a personal device can request that the Space Broker maintain continuous control over a characteristic's value at a specific location (see `characteristic.maintain` in Table 1). For example, a user can ask the Space Broker to maintain 700 lumens over their desk throughout the day. During most of the day (and depending on weather conditions), a significant amount of luminance may come from the office window. As conditions outside the window change, the Space Broker repeatedly updates the settings of the available office lights to balance the illumination level with sunlight's luminosity.

Interacting with characteristics in a smart space demands frequent reference to locations, which makes localization an essential capability of the space. Within the generic specification of the Space Broker, we do not embed a single definition of location. Rather a variety of localization options are easily integrated with the Space Broker model. For instance, the Space Broker (and hence the Space Broker proxy) may connect to a service provided in the space that allows the user to download a map (e.g., a floorplan) of the space and then use this map to select a location. Alternatively, a space may provide semantic labels for the space, and users may be able to use voice commands to indicate a specific location in the space. The references in Table 1 to `location` use the space's semantic meaning of location as provided by the localization service depicted in Figure 2. Finally, we have so far assumed that users are static, but if the user makes a request to the Space Broker to maintain the value of a characteristic at the user's location, even as the user moves, this can be accomplished with the Space Broker proxy by enabling the user to provide a reference to a user-side *locator* object resident on the personal device (as shown in Figure 2) in place of providing a concrete `location`. In this way, the user delegates responsibility for providing the needed `location` value to the `maintain` method to an object that dynamically fills in the value as the user moves, in collaboration with the space's localization service.

Resolving users' requests is a tall task for the Space Broker. The API presented in Table 1 simplifies the programming burden on the user side at the expense of embedding the complexity in the Space Broker. The ability to provide service for a particular characteristic depends on the presence of the right IoT devices and sensors in the space, upon knowledge about the histories of (successful and unsuccessful interactions by users in this space), models of the physics of the characteristics in the space, learning from user feedback, etc. The Space Broker's separation of concerns shields application programmers from any idiosyncracies of specific device manufacturers. In fact, because applications are written at the level of spatial characteristics, integrating new devices from new manufacturers has no impact on existing applications. In contrast, new devices will need to be integrated into the Space Broker, which requires coding an adaptor that maps the device's API onto the characteristic(s) that the device impacts. In the next section, we describe a first implementation of the Space Broker in a prototypical smart space. This implementation focuses on feasibility rather than on optimality, leaving the latter for future research.



(a) Schematic floorplan of the box. Yellow stars represent LEDs; red circles represent light sensors. The green ball represents the user.



(b) Overhead photograph of the box.

Figure 3: The Prototypical Space.

5 THE SPACE BROKER IN ACTION

Due to the lack of access to our laboratory facilities and real human participants, we built a small-scale experimental test environment using foam core walls, LED lights, LDR sensors, a cooling fan, and temperature sensors. The human participant is represented by an avatar placed in the space, a green ball that can be repositioned at will; to localize the personal device, we rely on an overhead camera that can track this ball. In contrast, we assume physical devices are fixed in the space. In real smart environments, devices may be added to or removed from the space, or they may be relocated within the same space. Our prior work [9] showed that, with respect to localizing such new devices, these updates can be accomplished with minimal user involvement. In the future, we will integrate more adaptive localization mechanisms with the Space Broker architecture. But for now, rather than using a localization service for the physical devices, we assume the positions of the physical devices are known to the Space Broker. To avoid conflicting uses of the devices at the user level, we only consider a single user in the space; handling conflicts among users is reserved for future work.

Figure 3 shows our prototypical smart space, labeling each component used and depicting how the space is divided into smaller spaces. The physical space was constructed using foam core panels. The interior walls are not permanently attached to allow the

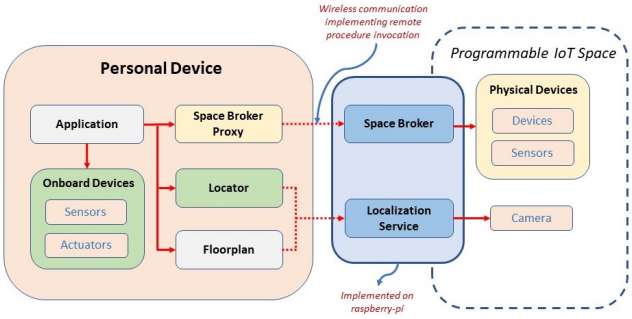


Figure 4: Concrete architecture of prototype Space Broker.

flexibility to change the layout and perform a wide range of experiments with minimal reconfiguration. Using this environment and its embedded sensors and actuators, applications can engage the Space Broker to query and control two characteristics: illumination and temperature. Light sensors and LEDs were used for any actions related to the illumination characteristic. A pair of LEDs (value ranging between 0 to 100 units) and an LDR sensor (capable of sensing value ranging between 0 to 1000 units) was located at each corner of the box. To avoid congestion in the picture, devices used for the temperature characteristic were not shown in the figures.

The Space Broker is implemented on a Raspberry-Pi. User applications run on an Android smart phone, which connects wirelessly to the Space Broker. To demonstrate the Space Broker API, we built an Android application that allows a user to query and control illumination and temperature. Even though the physical devices are fixed in the space and their locations are known to the Space Broker, we implemented a simple localization service to track the location of the "user" (e.g., green ball). The localization service is implemented on the same Raspberry-Pi device. It assumes a 2D coordinate system overlaid on the floorplan in Figure 3a.

Figure 4 depicts the architecture of the Space Broker as realized for this demonstration. The *Locator* object on the personal device is used by the application to obtain the user's location from the localization service. Similarly, the *Floorplan* object on the personal device is used by the application to retrieve the 2D map of the space from the localization service so that user's requests to the Space Broker can reference specific locations. The localization service has access to an overhead camera that was set up atop the foam core box with a top view of the entire space. We assume that every pixel in the camera frame corresponds to a coordinate. The primary purpose of the overhead camera is to detect the user's location. The secondary purpose is to capture the top view image of the space and send it to the personal device so that the user can click on any spot of the image to indicate a specific location in the space.

Figure 5 shows the GUI of our application, which supports querying, modifying, and maintaining the value of a characteristic either at the user location or at a location provided by the user.

Clicking on the *User Location* button in the Home Screen takes the user to a screen where they can query, modify, or maintain the value of a characteristic (in this example, illumination) at their location. Querying the value of illumination causes a `characteristic.query` action on the Space Broker proxy, which is in turn delivered to the Space Broker. The Space Broker uses

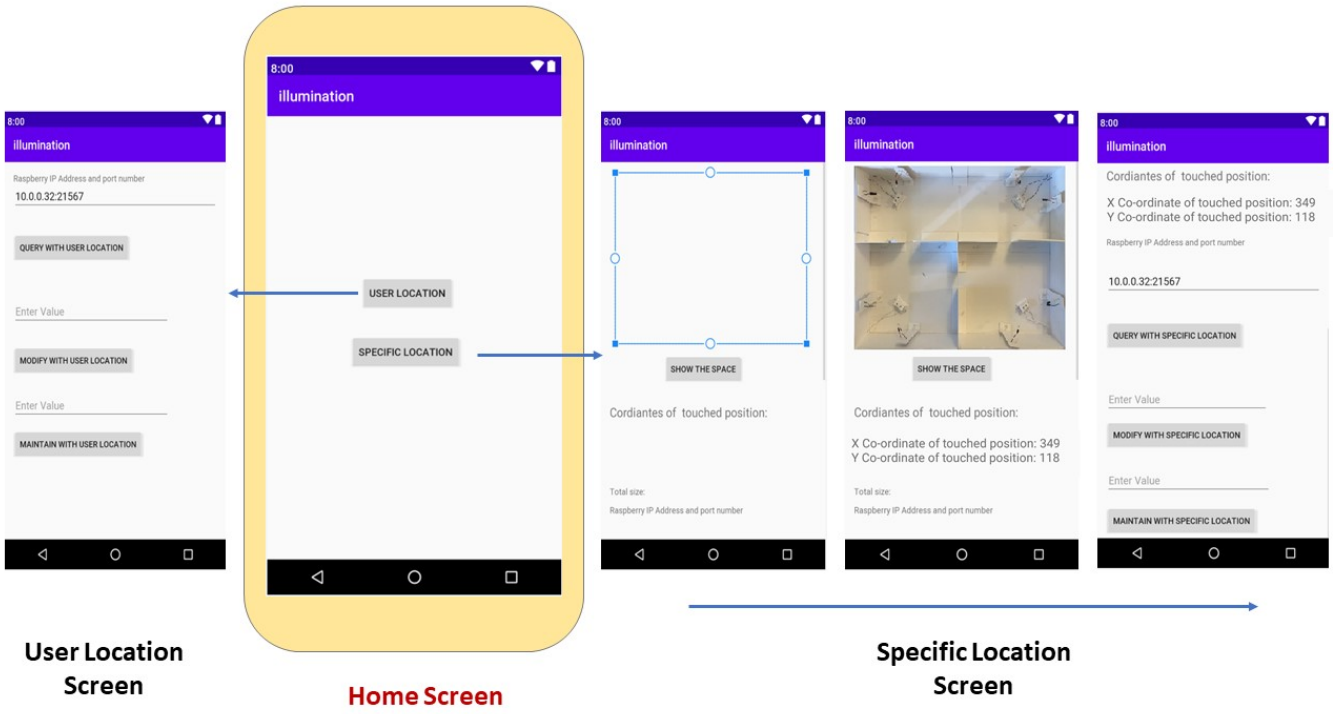


Figure 5: Screenshots of the Space Broker Application.

the provided location to identify the appropriate sensor to use (in this example, we simply use the closest sensor). Upon its return to the Android device, the queried result is displayed on the screen. Requesting to modify or maintain the characteristic value at the user's location requires the user to provide a desired value in the input fields before submitting the request. For each action, the user's location is automatically retrieved from the localization service before the request is sent to the Space Broker proxy.

To initiate user requests at a location *other* than the user's location, the user needs to enter the *Specific Location* for the request. In the Specific Location Screen, a user can request to display the top view of the 2D space. Then, by tapping on the displayed image of the space, the user can set the location they want to query, modify, or maintain. After selecting the desired location (e.g., x=349, y=118 as shown in Figure 5), the user can submit a request to the proxy.

Figure 6 depicts the sequence of actions that occur behind the scenes on Space Broker when a user initiates a request from the application. Suppose a user desires to achieve 850 units of illumination at their location. When they put the value of 850 as an input in the User Location Screen and submit a modify request, both the user location and the desired value are passed from the application to the Space Broker via the proxy. As a result, the LEDs around the user become operational, seeking to obtain 850 unit of luminosity.

This tabletop environment is a faithful implementation of the actual Space Broker, with an application that uses the provided API directly. It provides the ability to implement and measure real applications with real devices. Of course, such an environment has its limitations, especially with respect to not being able to perform a large number of experiments in diverse settings. A key avenue for future research is to integrate our implementation of the Space Broker with a smart space simulator to enable the additional benefit of measurements at different scales and in diverse spaces. This will become increasingly important as we implement multiple algorithms to resolve Space Broker requests and need to compare their performance in terms of accuracy and overhead.

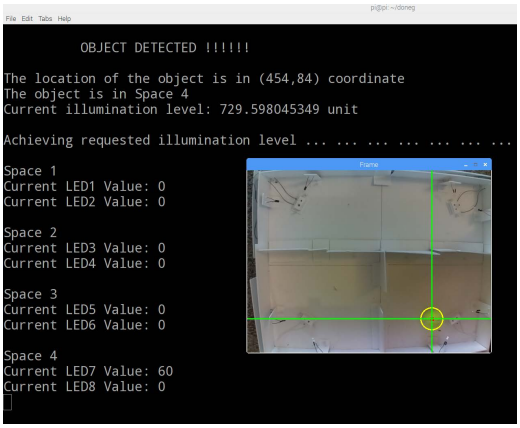
6 EVALUATION

Through our initial study with the test environment, we seek to answer three questions about the approach:

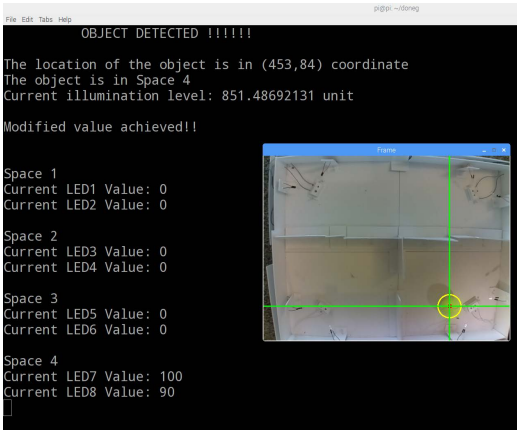
(1) What are the key novelties of the Space Broker in the context of programmable IoT smart spaces?
(2) What are the impacts on ease of programming when controlling *characteristics* rather than individual *devices*?
(3) Is the Space Broker API feasible in the context of a programmable smart space?

In this section, we use examples related to illumination to show the ease of programming and feasibility of implementing the Space Broker. The same arguments extend to other characteristics.

Novelty. It is not possible to measure novelty quantitatively, but we start our evaluation with a qualitative assessment of the Space Broker in the context of other models for interacting with programmable smart spaces. Existing available frameworks to interact with programmable IoT environments perceive the space as a collection of smart gadgets that are mostly accessed at the device level with the help of a mediating agent. For example, a user can request Alexa to turn on a light, but they cannot ask Alexa to impact the environment by requesting a specific level of illumination at a specific



(a) In this example, LED7 and LED8 are activated, attempting to achieve 850 lumens at the user location.



(b) The result is that 850 lumens is achieved at the user location when LED7 is at its maximum (100), and LED8 has a value of 90.

Figure 6: A terminal window shows the background activity of the Space Broker upon receiving a `modify` request for 850 lumens at the user's location.

location and expect Alexa to come up with a solution by controlling the right light sources in the space. One reason is that the existing middleware do not have an interface designed for taking user requests in the form of *characteristics* of the environment.

Like the leading existing interaction frameworks (such as Alexa, Google Assistant, or Siri), the Space Broker can act as a mediator between the user and a space, with the key novelty being the ability to accept user requests in terms of spatial characteristics. Further, all of the IoT devices of a smart environment are also within the Space Broker's awareness so that it can have full ability to set combinations of devices to achieve a user's desired effect.

The Space Broker allows user applications to observe the space, not as an assemblage of devices that can be managed, but as a set of spatial characteristics that can be identified and regulated. Not all smart devices (e.g., light sources) related to a single characteristic (e.g., illumination) can produce the same level of intensity (e.g., luminance). For this reason and others, different devices present

Listing 1: Modifying illumination using Space Broker API

```
1  //... inside the Android activity
2  Characteristic illumination = // ... retrieved from Space Broker proxy
3  btnModifyUserLocation.setOnClickListener(new View.OnClickListener(){
4      @Override
5      public void onClick(View v){
6          String modifyValue = modifyValueUserLocation.getText().toString();
7          illumination.modify(null, modifyValue);
8      }
9  });
```

Listing 2: Modifying illumination using devices directly

```
1  //... inside the Android activity
2  Device[] lights = // ... retrieved via some discovery process
3  Device[] lightSensors = // .. retrieved via some discovery process
4  btnModifyUserLocation.setOnClickListener(new View.OnClickListener(){
5      @Override
6      public void onClick(View v){
7          Location loc = Locator.getLocation()
8          String modifyValue = modifyValueUserLocation.getText().toString();
9          Map<Location, Integer> lightLevels = senseLight(loc, lightSensors);
10         Map<Device, Integer> targetLevels =
11                 determineSettings(loc, lights, lightLevels, modifyValue);
12         targetLevels.forEach((k,v) -> k.setValue(v);
13     }
14 });
```

dramatically different control interfaces to the user. To use an overhead light, a user might employ a dimmer switch mounted on the wall or a slider in the manufacturer's smartphone app. In contrast, to adjust the light level by raising or lowering the window shades, the user may employ a cord on the shades or a raise-and-lower function in the manufacturer's application. Each of these controls may have widely varying impacts on the level of illumination. However, by unifying these controls under the single abstraction of *characteristic*, the Space Broker provides a novel way for applications to view a smart space. We posit that this will in turn ease the burden of programming smart spaces.

Ease of Programming. To present the Space Broker's impact on easing the programming smart spaces, we examine a snippet of code that relies on the Space Broker API (Listing 1). In particular, this code shows the implementation for modifying the illumination characteristic at the user's location.

In Listing 1, we show the listener associated with the `modify` button in the far left screenshot of Figure 5. Line 6 gets the user's desired illumination level from the user interface, and line 7 calls `modify` on the `illumination` characteristic that was created by the proxy when the user device connected to the Space Broker. In line 7, the first parameter indicates the location of the request; when it is set to `null`, the Space Broker proxy fills in the user's location before sending the request to the Space Broker. The proxy does this by interacting with the on-device *Locator*. In our prototype environment, this *Locator* in turn relies on the overhead camera that provides the position of the green ball relative to the space's coordinate space. Again, the specific semantics of location in the space are defined by the localization service, which is accessed by both the personal device's *Locator* and the Space Broker implementation.

In contrast, to implement this functionality without the assistance of the Space Broker abstraction, the user's application would be required to (1) discover all of the available illumination devices in the space; (2) assess the contextual conditions of the space before selecting one or more devices to control and (3) issue directives directly to these devices, which may provide diverse interfaces.

Listing 2 shows a sketch of a simplified version of this device-specific approach. This sketch assumes that all light devices have the same interface. As shown in lines 2-3, the application must discover the available devices without the support of the Space Broker (and rediscover them if the devices in the space change). When the user wishes to execute a request, the application must retrieve the user's location (line 7), retrieve the target value from the user interface (line 8), assess the current light levels (line 9), determine the needed levels for each light device (lines 10-11), and then iterate over the available lighting devices to change their settings (line 12). Lines 9-11 assume the availability of algorithms to assess the space and the devices it contains. In the Space Broker implementation, the application delegates these algorithms to the Space Broker rather than implementing them as part of the application. This allows the algorithms to belong to the space and the user's application to leverage different algorithms in different spaces. In this way, the application can dynamically leverage the availability of diverse devices as its user encounters differently configured spaces.

Feasibility. We have implemented the Space Broker, its proxy, and an example Android application, all as described in the previous section. To ensure wide reproducibility, we have made our code publicly available[1]. To give a sense of how the Space Broker Proxy realizes the characteristic-based interface, we show the implementation of a proxied characteristic in Listing 3.

The listing shows the API invoked by the example in Listing 1. The `illumination` object in Listing 1 is of type `Characteristic`, which means it implements the `CharacteristicInterface` to provide the `query`, `modify`, and `maintain` methods. These methods take the provided location (or resolve that to the user location if no location is provided) and the desired value (in the case of `modify` and `maintain`), create a String request, and dispatch that request to the Space Broker via the available wireless communication.

The Space Broker implementation is written in Python. It receives string requests from the proxy and uses the physical devices to resolve the requests. To implement a query, the current implementation simply finds the light sensor closest to the request's location and returns the value sensed at that location. To implement a modify request, the Space Broker starts by attempting to adjust the light nearest to the request's location. The Space Broker adjusts this light up or down depending on the difference between the target value and the sensed value at the nearest light sensor. If the target value can be achieved using only the nearest light, the Space Broker finishes and returns. Otherwise, the Space Broker continues to the next nearest light, and so on, until it either reaches the target value or exhausts all of its options. To implement a maintain request, the Space Broker performs the same actions as for modify, but it repeats these in a loop, continuously sensing the environment and adjusting as needed.

This implementation of the Space Broker, its API, and the proxy that delivers its API to application developers demonstrates the feasibility of implementing the *characteristic* abstraction and building real applications that rely on that abstraction rather than requiring developers to program to device interfaces. As exemplified through this study, the Space Broker brings several important benefits:

[1]https://github.com/HamimAdal/Middleware_Space_Broker

Listing 3: Implementation details for Space Broker Proxy

```
1  interface CharacteristicInterface { // Space Broker proxy API methods
2      String query(Location location);
3      void modify(Location location, String Value);
4      void maintain(Location location, String Value);
5  }

7  class Characteristic implements CharacteristicInterface {
8      // *****************
9      // STS is the string or message that will be sent to the space broker
10     // (on the raspberry-pi) from the application. It includes:
11     //    1. requestId and/or,
12     //    2. location and/or,
13     //    3. value.
14     // It is formatted as a string for ease of transmission and processing
15     // on the raspberry-pi side.
16     // *****************

18     @Override
19     public String query( Location location ) {
20         Location loc = resolveLocation( location );
21         String requestId = REQUEST_TYPES.QUERY;
22         String STS = requestId + ":" + loc.X + ":" + loc.Y ;
23         sendRequest(STS);

25         DataInputStream inp = new DataInputStream(socket.getInputStream());
26         queriedValue = inp.readUTF(); // receive queried value
27         return queriedValue;
28     }

30     @Override
31     public void modify( Location location, String value ) {
32         Location loc = resolveLocation( location );
33         String requestId = REQUEST_TYPES.MODIFY;
34         String STS = requestId + ":" + loc.X + ":" + loc.Y + ":" + value;
35         sendRequest(STS);
36     }

38     @Override
39     public void maintain( Location location, String value ) {
40         Location loc = resolveLocation( location );
41         String requestId = REQUEST_TYPE.MAINTAIN;
42         String STS = requestId + ":" + loc.X + ":" + loc.Y + ":" + value;
43         sendRequest(STS);
44     }

46     private void resolveLocation(Location location){
47         return (location == null) ? Locator.getUserLocation() : location;
48     }

50     private void sendRequest(String request){
51         wirelessCommunication wireless = new wirelessCommunication();
52         wireless.execute(STS); // sends string request to Space Broker
53     }
54 }
```

- By allowing applications to reason at the level of a *characteristic*, diverse device types that impact the same characteristic can be brought under the same control mechanism. For example, the settings of lights and window shades both affect illumination, and the application can delegate to the space how to adjust them in concert to realize a lighting goal.
- Users' policies and preferences related to smart spaces exist at the characteristic level, but current implementations are at the device level. By separating policies and preferences (on the application-side of the Space Broker API) from device-level implementations (within the Space Broker), our model makes user- and application-level settings transferable across spaces. It also allows different spaces to provide different implementations of these policies depending on their low level device capabilities.
- The ease of programming that comes with the Space Broker API allows application developers to distance themselves from concerns related to network connections and low-level device interfaces, resulting in code that is less error prone.

7 DISCUSSION AND FUTURE WORK

The Space Broker as described is a first demonstration of spatial characteristics and their capabilities. Future work will focus on both algorithms for resolving complex requests and mediating conflicts among multiple users in the space. Of particular interest is developing more robust models of the impacts of diverse devices on characteristics. For instance, the ways that window shades, fans, heating systems, and cooktops impact a space's temperature are diverse and vary with both time and space. Embedding an understanding of the physics of characteristics into the Space Broker would improve the quality of user interactions with the smart space. Similarly, requests from multiple users may result in conflicts that the Space Broker must negotiate. Immediately ongoing work on the Space Broker seeks to identify such conflicts and resolve them, through explicit user engagement as well as automatically.

Further, a user may have preferences with respect to possible solutions to a request (e.g., one user may prefer natural light, while another prefers to keep the shades closed for privacy). Future work will explore extending the Space Broker to allow for such preferences, both explicitly, as part of the API calls and implicitly, for instance as learned through feedback from users.

Another important area of future work is developing efficient algorithms to resolve users' requests in complex spaces. In relatively small spaces, we expect to be able to derive optimal algorithms; in more complex, unpredictable, or dynamic spaces, we expect to rely on heuristics. In addition, the Space Broker's algorithms are only as good as the quality of the information that the Space Broker has to act on. If sensors are of poor quality or report incorrect information, the Space Broker may also perform incorrectly. We plan to address these uncertainties and inconsistencies through redundant sensing (e.g., using sensors of different modalities for a single characteristic) and through user feedback. Our prior work demonstrated that it is feasible to collect feedback from users both explicitly and implicitly [9, 10]. In the latter case, it is possible to observe a series of user interactions (e.g., requesting a certain illumination level then immediately requesting a higher level) and infer that the space is not behaving in the manner the user expects. These observations can help detect inconsistencies in sensing and adjust the Space Broker algorithms to account for them.

Finally, the security of smart spaces is essential. On one hand, the Space Broker itself must be protected. It embeds knowledge a user's devices and sensors, the user's location, the floorplan of the home, and, ultimately, the user's behaviors and habits. Future work will ensure protection of the Space Broker, e.g., by encrypting and authenticating all interactions between devices and the Space Broker. On the other hand, the Space Broker can also be leveraged to help users control the physical security of their smart spaces, an area of research that we are currently exploring.

8 CONCLUSION

A framework for interacting with programmable smart spaces should be able to accommodate various types of spatial characteristics of the space that are distinct in nature. With the Space Broker interface, applications can specify useful interactions for examining and manipulating characteristics of the space like "illumination", "temperature", "sound" etc. While we focused in this paper on *scalar* characteristics, not all smart space characteristics fit this model. For instance, handling characteristics such as "security" or "streaming media" is a more challenging task. The concept of security can be conceptualized in a variety of ways, from monitoring the boundaries of the space, protecting a region within the space, detecting the presence of unknown persons, etc. To connect streaming media to streaming-capable devices in a space, a user needs to have the ability to change the media stream or to have an active stream follow them throughout the space. In the future, we will examine the Space Broker to accommodate such characteristics.

ACKNOWLEDGEMENTS

This work was funded in part by the National Science Foundation under grants CNS-1813263, CNS-1909221, CNS-1907959. Any opinions, findings, conclusions or recommendations expressed are those of the authors and do not necessarily reflect the views of the NSF.

REFERENCES

[1] Adhatarao, S.S., Arumaithurai, M., Kutscher, D., Fu, X.: Isi: Integrate sensor networks to internet with icn. IEEE Internet of Things Journal **5**(2), 491–499 (2017)
[2] Amadeo, M., Campolo, C., Iera, A., Molinaro, A.: Information centric networking in iot scenarios: The case of a smart home. In: 2015 IEEE international conference on communications (ICC). pp. 648–653. IEEE (2015)
[3] Ascigil, O., Reñé, S., Xylomenos, G., Psaras, I., Pavlou, G.: A keyword-based icn-iot platform. In: Proceedings of the 4th ACM Conference on Information-Centric Networking. pp. 22–28. ACM (2017)
[4] Bracciale, L., Loreti, P., Detti, A., Paolillo, R., Melazzi, N.B.: Lightweight named object: an icn-based abstraction for iot device programming and management. IEEE Internet of Things Journal (2019)
[5] Braun, T., Hilt, V., Hofmann, M., Rimac, I., Steiner, M., Varvello, M.: Service-centric networking. In: 2011 IEEE International Conference on Communications Workshops (ICC). pp. 1–6. IEEE (2011)
[6] Eisenhauer, M., Rosengren, P., Antolin, P.: Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems. In: The Internet of Things, pp. 367–373 (2010)
[7] Felemban, E., Murad, M., Manzoor, M.A., Sheikh, A.A.: Unigate: Modular universal wireless gateway. In: 2014 World Congress on Computer Applications and Information Systems (WCCAIS). pp. 1–3. IEEE (2014)
[8] Honkola, J., Laine, H., Brown, R., Tyrkkö, O.: Smart-m3 information sharing platform. In: The IEEE symposium on Computers and Communications. pp. 1041–1046. IEEE (2010)
[9] Hua, J., Lee, S., Roman, G.C., Julien, C.: Arciot: Enabling intuitive device control in the internet of things through augmented reality. In: 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops). pp. 558–564. IEEE (2021)
[10] Hua, J., Liu, C., Kalbarczyk, T., Wright, C., Roman, G.C., Julien, C.: riot: Enabling seamless context-aware automation in the internet of things. In: Proceedings of the 16th international conference on Mobile Ad-hoc and Smart Systems. pp. 227–235. IEEE Press (2019)
[11] Korzun, D.G., Kashevnik, A.M., Balandin, S.I., Smirnov, A.V.: The smart-m3 platform: Experience of smart space application development for internet of things. In: Internet of Things, Smart Spaces, and Next Generation Networks and Systems, pp. 56–67. Springer (2015)
[12] Morandi, F., Roffia, L., D'Elia, A., Vergari, F., Cinotti, T.S.: Redsib: a smart-m3 semantic information broker implementation. In: 2012 12th Conference of Open Innovations Association (FRUCT). pp. 1–13. IEEE (2012)
[13] Nath, S.: Ace: exploiting correlation for energy-efficient and continuous context sensing. In: Proc. of MobiSys. pp. 29–42 (2012)
[14] Perera, C., et al.: Context aware computing for the internet of things: A survey. IEEE Communications Surveys & Tutorials **16**(1) (2014)
[15] Saputra, Y., Hua, J., Wendt, N., Julien, C., Roman, G.C.: Warble: programming abstractions for personalizing interactions in the internet of things. In: Proceedings of the 6th International Conference on Mobile Software Engineering and Systems. pp. 128–139. IEEE Press (2019)
[16] Yus, R., Bouloukakis, G., Mehrotra, S., Venkatasubramanian, N.: Abstracting interactions with iot devices towards a semantic vision of smart spaces. In: Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation. pp. 91–100 (2019)
[17] Zhang, Y., Raychadhuri, D., Ravindran, R., Wang, G.: Icn based architecture for iot. IRTF contribution, October (2013)