Supporting User Interactions with Smart Built Environments

Mohamed Hussein Hafez Handosa

Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Computer Science and Application

Denis Gračanin, Chair
Hicham G. Elmongui, Co-Chair
Aisling Kelliher
Benjamin Knapp
Krešimir Matković

December 10, 2018
Blacksburg, Virginia

Keywords: Human-Computer Interaction, Internet of Things, Smart Environments

Copyright 2018, Mohamed H. Handosa
Supporting User Interactions with Smart Built Environments

Mohamed Hussein Hafez Handosa

ABSTRACT

Before the recent advances in sensing, actuation, computing and communication technologies, the integration between the digital and the physical environment was limited. Humans linked those two worlds by collecting data about the physical environment before feeding it into the digital environment, and by changing the state of the physical environment based on the state of the digital environment.

The incorporation of computing, communication, sensing, and actuation technologies into everyday physical objects has empowered the vision of the Internet of Things (IoT). Things can autonomously collect data about the physical environment, exchange information with other things, and take actions on behalf of humans. Application domains that can benefit from IoT include smart buildings, smart cities, smart water, smart agriculture, smart animal farming, smart metering, security and emergencies, retail, logistics, industrial control, and health care.

For decades, building automation, intelligent buildings, and more recently smart buildings have received considerable attention in both academia and industry. We use the term smart built environments (SBE) to describe smart, intelligent, physical, built, architectural spaces ranging from a single room to a whole city. Legacy SBEs were often closed systems operating their own standards and custom protocols. SBEs evolved to Internet-connected systems leveraging the Internet technologies and services (e.g., cloud services) to unleash new capabilities. IoT-enabled SBEs, as one of the various applications of the IoT, can change the way we experience our homes and workplaces significantly and make interacting with technology almost inevitable. This can provide several benefits to modern society and help to make our life easier. Meanwhile, security, privacy, and safety concerns should be addressed appropriately.
Unlike traditional computing devices, things usually have no or limited input/output (I/O) capabilities. Leveraging the ubiquity of general-purpose computing devices (e.g., smartphones), thing vendors usually provide interfaces for their products in the form of mobile apps or web-based portals. Interacting with different things using different mobile apps or web-based portals does not scale well. Requiring the user to switch between tens or hundreds of mobile apps and web-based portals to interact with different things in different smart spaces may not be feasible. Moreover, it can be tricky for non-domestic users (e.g., visitors) of a given SBE to figure out, without guidance, what mobile apps or web-based portals they need to use to interact with the surrounding.

While there has been a considerable research effort to address a variety of challenges associated with the thing-to-thing interaction, human-to-thing interaction related research is limited. Many of the proposed approaches and industry-adopted techniques rely on more traditional, well understood and widely used Human-Computer Interaction (HCI) methods and techniques to support interaction between humans and things. Such techniques have mostly originated in a world of desktop computers that have a screen, mouse, and keyboard. However, SBEs introduce a radically different interaction context where there are no centralized, easily identifiable input and output devices. A desktop computer of the past is being replaced with the whole SBE. Depending on the task at hand and personal preferences, a user may prefer to use one interaction modality over another. For instance, turning lights on/off using an app may be more cumbersome or time-consuming compared to using a simple physical switch.

This research focuses on leveraging the recent advances in IoT and related technologies to support user interactions with SBEs. We explore how to support flexible and adaptive multimodal interfaces and interactions while providing a consistent user experience in an SBE based on the current context and the available user interface and interaction capabilities.
Supporting User Interactions with Smart Built Environments

Mohamed Hussein Hafez Handosa

GENERAL AUDIENCE ABSTRACT

The recent advances in sensing, actuation, computing, and communication technologies have brought several rewards to modern society. The incorporation of those technologies into everyday physical objects (or things) has empowered the vision of the Internet of Things (IoT). Things can autonomously collect data about the physical environment, exchange information with other things, and take actions on behalf of humans. Several application domains can benefit from the IoT such as smart buildings, smart cities, security and emergencies, retail, logistics, industrial control, and health care.

For decades, building automation, intelligent buildings, and more recently smart buildings have received considerable attention in both academia and industry. We use the term smart built environments (SBE) to describe smart, intelligent, physical, built, architectural spaces ranging from a single room to a whole city. SBEs, as one of the various applications of the IoT, can change the way we experience our homes and workplaces significantly and make interacting with technology almost inevitable.

While there has been a considerable research effort to address a variety of challenges associated with the thing-to-thing interaction, human-to-thing interaction related research is limited. Many of the proposed approaches and industry-adopted techniques to support human-to-thing interaction rely on traditional methods. However, SBEs introduce a radically different interaction context. Therefore, adapting the current interaction techniques and/or adopting new ones is crucial for the success and wide adoption of SBEs.

This research focuses on leveraging the recent advances in the IoT and related technologies to support user interactions with SBEs. We explore how to support a flexible, adaptive, and multimodal interaction experience between users and SBEs using a variety of user interfaces and proposed interaction techniques.
To Sahar, Ziad, and Malik
Acknowledgments

I cannot find the words to express my gratitude to my advisor Dr. Denis Gračanin and my co-advisor Dr. Hicham G. Elmongui for their invaluable support throughout my Ph.D. research. They have taught me a lot and this Ph.D. work would not have been possible without their encouragement and support.

I would like to thank my committee members, Dr. Aisling Kelliher, Dr. Benjamin Knapp, and Dr. Krešimir Matković for their guidance that has helped in shaping the direction of my research work. I appreciate their cooperation and dedication to support this work since our first meeting.

I am very thankful to my lab mates and colleagues in the Department of Computer Science. I have been fortunate to have them around me, especially the wonderful Archi Dasgupta, Mark Manuel, Reza Tasooji, Andrew Ciambrone, and Matthew Tucker. I would like as well to thank the FutureHAUS team, that I had the honor to be part of it, for the great experience and the remarkable achievements.

My sincere gratitude to the VT-MENA program director, Dr. Sedki Riad, for his endless support and invaluable guidance. The support of my friends in Virginia Tech and around Blacksburg has been always exceptional. They have always been there for me no matter what this will cost them.

Finally, my sincere appreciation to my parents, sister, wife, and kids. Without their support, I could never have got this work done. Thank you, my wonderful family.
Contents

1 Introduction .................................................. 1
  1.1 User Interaction with SBEs ................................. 3
  1.2 Motivation .................................................. 6
    1.2.1 Why CPS? ............................................. 7
    1.2.2 Why Explicit Interaction? ............................ 8
    1.2.3 Why Multimodal Interaction? ......................... 8
    1.2.4 Why Not Vendor-Provided Apps? ..................... 11
  1.3 Methodology ............................................... 12
  1.4 Contribution ................................................. 15
  1.5 Document Organization .................................... 16

1 Background and Related Work .............................. 17

2 The Internet of Things ...................................... 18
  2.1 IoT Enabling Technologies ................................. 21
    2.1.1 Wireless Communication ............................. 21
5 Proposed Approach

5.1 Thing/Service Information

5.1.1 SBE Ontology

5.1.2 Registration Service

5.2 Discovering an SBE

5.3 Learning about an SBE

5.3.1 Learning Service

5.3.2 Thing/Service Description

5.4 Generating an SBE UI

5.5 Adapting an SBE UI

5.6 Maintaining UI/SBE Consistency

III Evaluation and Discussion

6 Simulating MQTT-Based Communication

7 Supporting User Interaction

7.1 Gesture-Based Light Control

7.1.1 Approach
List of Figures

2.1 The IoT-Lite ontology [Bermudez-Edo et al., 2017] ......................... 67


4.1 The four form dimensions of an interaction proxy. ......................... 86

4.2 An example of a special-purpose interaction proxy. ....................... 87

4.3 Functionality distribution across things in an SBE compared to traditional isolated devices in a typical house setup. ......................... 92

5.1 A framework to describe explicit user interaction in SBEs. ............... 102

5.2 A four-step process to automatically generate a UI for an SBE. ........ 103

5.3 Autonomous UI generation framework. ................................... 105

5.4 An overview of the SBE ontology. ....................................... 108

5.5 Thing/service registration service. ..................................... 111

5.6 Discovering SBEs using a discovery service. ............................. 112

5.7 Learning about an SBE using a learning service. ....................... 114

5.8 Generating a UI from thing/service descriptors. ....................... 118
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>The publish/subscribe communication pattern of the MQTT protocol.</td>
<td>123</td>
</tr>
<tr>
<td>6.2</td>
<td>Virtual machines connected through a simulated network.</td>
<td>124</td>
</tr>
<tr>
<td>6.3</td>
<td>Results obtained from the MQTT simulation tool.</td>
<td>125</td>
</tr>
<tr>
<td>7.1</td>
<td>A real-world example of a fairly complex set of light switches and regulators.</td>
<td>128</td>
</tr>
<tr>
<td>7.2</td>
<td>(a) Philips Hue app, (b) Philips Hue Tap kinetic powered switch.</td>
<td>130</td>
</tr>
<tr>
<td>7.3</td>
<td>A taxonomy of basic light control tasks.</td>
<td>131</td>
</tr>
<tr>
<td>7.4</td>
<td>Selecting a light source.</td>
<td>134</td>
</tr>
<tr>
<td>7.5</td>
<td>Lighting control system.</td>
<td>135</td>
</tr>
<tr>
<td>7.6</td>
<td>The lighting control system deployed in a living room.</td>
<td>136</td>
</tr>
<tr>
<td>7.7</td>
<td>Video recording frames for selecting a light source.</td>
<td>137</td>
</tr>
<tr>
<td>7.8</td>
<td>A thing annotated with MR-based UI elements.</td>
<td>142</td>
</tr>
<tr>
<td>7.9</td>
<td>Air-tap gesture, a switch from the ready to the press state. [Microsoft, 2016]</td>
<td>144</td>
</tr>
<tr>
<td>7.10</td>
<td>Integrating an SBE-side tracking device with a user-side MR headset to form a multiple-device Interaction Proxy.</td>
<td>146</td>
</tr>
<tr>
<td>7.11</td>
<td>Collecting a point-pair from the Kinect device and the HoloLens device.</td>
<td>147</td>
</tr>
<tr>
<td>7.12</td>
<td>A registered skeleton aligned with the corresponding physical body.</td>
<td>148</td>
</tr>
<tr>
<td>7.13</td>
<td>Two alternative communication models: (a) UDP-based direct communication and (b) MQTT-based indirect communication.</td>
<td>148</td>
</tr>
<tr>
<td>7.14</td>
<td>(a) The virtual hospital room. (b) Demonstration of using both hands simultaneously for brushing a denture.</td>
<td>149</td>
</tr>
<tr>
<td>7.15</td>
<td>Video recording consecutive frames for detecting a closed hand gesture using Kinect and providing feedback through HoloLens.</td>
<td>151</td>
</tr>
</tbody>
</table>
7.16 A closed hand (or press) gesture recognized by both Kinect and HoloLens.  . 152

8.1 Layout of the study equipment.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155

8.2 User study equipment setup.  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 157

8.3 Average task completion time in round 1 (learnability).  . . . . . . . . . . . . 160

8.4 Average task completion time in round 2 (memorability).  . . . . . . . . . . . 161

8.5 Average task completion time in round 3 (efficiency).  . . . . . . . . . . . . 162

8.6 SUS scores box plot of the UIs for Session 1.  . . . . . . . . . . . . . . . . . . . 163

8.7 SUS scores box plot of the UIs for Session 2.  . . . . . . . . . . . . . . . . . . . 164
List of Tables

2.1 Comparison of discovery services ........................................ 62
3.1 User interaction models in IoT-enabled SBEs .......................... 73
3.2 Levels of interactivity in context-aware applications .................. 74
3.3 UI usability comparison for different input types ..................... 78
3.4 UI usability comparison for different output types ........................ 79
4.1 Input type support in different interaction proxy forms ................ 88
4.2 Output types support in different interaction proxy forms .......... 89
5.1 IoT-Lite ontology support for positioning information .............. 107
5.2 The SBE ontology data-type properties ......................... 109
8.1 Demographics of the participating subjects ......................... 159
8.2 Experience of the participating subjects ................................. 159
8.3 The average SUS scores for each of the UIs in both sessions .... 163
8.4 The analysis of variance (ANOVA) results for the UIs in both sessions ... 164
Chapter 1

Introduction

Sensing and actuation technologies have brought several rewards to modern society. Leveraging those technologies, several academic and industrial efforts have evolved significantly over the past years. Before the recent advances in sensing and actuation technologies, the integration between the digital world and the physical world was limited. Humans used to link both worlds by (1) collecting data about the physical environment before feeding it into the various digital systems and (2) changing the state of the physical environment based on the outputs of those digital systems. Incorporating sensing and actuation capabilities into the physical environment creates a bridge that allows for better integration between the digital and the physical worlds, enabling them to interact autonomously. Sensors can collect data about the physical world before feeding it into the digital world and actuators allow the digital world to change the state of the physical world.

The incorporation of processing, communication, sensing, and actuation technologies into everyday physical objects has empowered the vision of the Internet of Things (IoT). The term “thing” refers to a uniquely identifiable object with a physical embodiment that is equipped with computing and communication as well as sensing and/or actuation capabilities [Miorandi et al., 2012]. Traditionally, the Internet has mostly focused on supporting human-to-human interaction through interconnected computing devices that enabled infor-
mation exchange over the network. However, the incorporation of things into the Internet has enabled new interaction paradigms, namely, thing-to-thing and human-to-thing interaction [Ortiz et al., 2014]. Things can autonomously collect data, exchange information with other things, and take actions on behalf of humans. Several application domains can benefit from the IoT including smart cities, smart environment, smart water, smart metering, security and emergencies, retail, logistics, industrial control, smart agriculture, smart animal farming, smart buildings, and health care [Asin and Gascon, 2012].

The IoT introduces a disruptive level of innovation that can change our world dramatically [Atzori et al., 2010]. It provides unprecedented opportunities to penetrate technology into our daily life. Unlike the traditional computing systems, where humans had the choice to either use or avoid using them, the IoT will make interacting with technology almost inevitable. As the IoT continues to grow and the number of things around us continues to increase, we will be interacting with technology whether cognitively or spontaneously. This can provide several benefits and help to make our life easier. Meanwhile, security, privacy, and safety concerns should be addressed appropriately.

The IoT forms a foundation for cyber-physical systems. “Cyber-Physical System (CPS) are physical and engineered systems whose operations are monitored, coordinated, controlled and integrated by a computing and communication core” [Rajkumar et al., 2010]. The combination of the IoT, artificial intelligence, machine learning, and cloud computing empowers the CPS revolution, where physical and computational components are deeply intertwined providing systems that can interact with human beings, learn about them, and adapt to their needs. CPS can play a key role in building progression. Newly constructed as well as existing buildings can incorporate IoT devices to serve as a foundation for providing CPS that can contribute to improving their capabilities by turning them into smart buildings.

Building automation, intelligent buildings, and smart buildings are three terms that are used extensively in the literature to refer to progression in buildings. In [Buckman et al., 2014], the authors have identified three main drivers of building progression (efficiency, longevity,
and comfort) and four methods (intelligence, control, enterprise, and material and construction). Building automation aims at minimizing user interaction while intelligent buildings focus on gathering information and responding to it autonomously. More recently, the term “smart buildings” is gaining popularity in the literature. It refers to buildings that account for not only intelligence but also enterprise, control, and materials and construction [Buckman et al., 2014]. A number of proposed designs and prototypes for building automation, intelligent buildings, and smart buildings can be found in the literature [Wong et al., 2005, Alaa et al., 2017].

We use the term Smart Built Environments (SBE) to describe smart, intelligent, controllable, cyber-physical, built, architectural spaces. Legacy systems for building automation, intelligent buildings, and smart buildings were often closed systems operating their own standards and custom protocols. Leveraging the IoT technologies, those systems can evolve from being isolated systems into Internet-connected systems that can benefit from the Internet technologies and services (e.g. cloud services) to unleash new capabilities that were not achievable in the past. IoT-enabled SBEs, as one of the various applications of the IoT, can dramatically change the way we experience our homes and workplaces.

1.1 User Interaction with SBEs

A user may interact with an SBE implicitly or explicitly. In implicit user interaction, the user interacts with the SBE spontaneously. For instance, an SBE may detect user presence via a motion sensor and autonomously turn the lights on. In contrast, explicit user interaction refers to intended interaction, where the user cognitively interacts with the SBE. For instance, a user may turn the lights on using a smartphone app. An SBE relies on its intelligence to support implicit user interaction by taking autonomous decisions based on the current context with no explicit input from its users. Meanwhile, it relies on its control support to enable explicit user interaction by providing its users with the appropriate UIs.
Typically, an SBE incorporates different things provided by different vendors. Those things come in various form factors with varying functionalities and I/O capabilities. Thing vendors often rely on well-established technologies to support user interaction with their products. They provide software-based UIs that usually take the form of mobile apps or web-based portals, allowing users to interact with things through smartphones and other general-purpose computing devices. This provides a cost-effective and flexible alternative compared to incorporating fully functional UIs into the things themselves. Thereby, thing vendors can decrease the hardware production cost and maintain the UIs through software updates rather than issuing product recalls. Moreover, users can interact with things remotely. However, besides those rewards, there are several challenges, as listed below. Those challenges can degrade the user experience if not addressed appropriately.

- **Consistency.** Decoupling things and their UIs can cause inconsistencies. A thing may have several UIs running on different devices. Ideally, a change in one UI should quickly propagate to the thing itself as well as the other UIs. Consequently, a user can switch seamlessly between different thing UIs and still experience a fluent interaction.

- **Interoperability.** Due to the lack of IoT standardization, it is usually challenging to achieve interoperability between things provided by different vendors. Therefore, users usually interact with things using custom vendor-provided apps. Currently, some smartphone apps (e.g. Samsung SmartThings, Belkin WeMo) allow their users to interact with different things provided by different vendors but their list of supported things is still limited to specific products.

- **Inter-usability.** An SBE should provide its users with coherent thing UIs rather than disjoint heterogeneous UIs. Coherent UIs should have consistent conceptual models, functionality organization, and interaction logic as well as consistent terminologies and visual designs if applicable. This can be challenging in an SBE that incorporates things provided by different vendors because custom vendor-provided UIs usually reflect varying insights.
• **Distributed functionality.** A functionality can be distributed across multiple things and a thing can be utilized by different services to provide different functionalities. As the number of things and supported services increases in an SBE, the interrelationships between things and services become more complex. Consequently, it becomes more challenging for the user to understand those relationships and to control things individually.

• **Conflict resolution.** An SBE can have multiple users with conflicting needs. Moreover, user needs may conflict with some SBE goals (e.g., reducing power consumption). The SBE should utilize its intelligence to address those conflicts appropriately.

• **Usability.** As the number of things increases in an SBE, the number of custom vendor-provided UIs will increase as well. This will require the user to learn and memorize numerous UIs, which can degrade the usability of the SBE in terms of learnability and memorability. Moreover, manually switching between different UIs to accomplish different tasks requires more interactions, which can degrade the usability of the SBE in terms of efficiency, error frequency, and user satisfaction.

• **Context awareness.** A context-aware UI should facilitate user tasks by autonomously adapting to the current context, thereby reducing the required interactions. However, failing to infer the current context correctly can result in inappropriate UI adaptations that can complicate the user experience rather than simplifying it.

One of the main goals of an SBE is to facilitate user tasks and to make the lives of its occupants more comfortable. Poor user interaction support can violate this goal regardless of the capabilities of the SBE. For SBEs to gain popularity, they need to provide not only useful functionalities but also convenient user interaction techniques. Therefore, addressing the challenges mentioned above is crucial for the success and wide adoption of SBEs.

The current trend in both academia and industry is to leverage well-established HCI techniques to support user interaction with SBEs. However, those techniques can introduce
several interaction limitations when deployed in SBEs. Researchers can help address those limitations by exploring the context of SBEs and studying how it differs from traditional computing before proposing solutions to improve user interaction in SBEs. Such solutions may involve adapting the current interaction techniques to perform better in the context of SBEs or adopting new techniques that, by design, consider the characteristics, the constraints, and the requirements associated with supporting user interaction with SBEs.

1.2 Motivation

While there has been a considerable research effort to address a variety of challenges associated with thing-to-thing interaction, human-to-thing interaction related research is limited. Many of the proposed approaches and industry-adopted techniques rely on legacy Human-Computer Interaction (HCI) methods to support interaction between humans and things. Perhaps thing vendors are trying to benefit from the available well-established and widely adopted technologies to provide cost-efficient user interfaces (UIs) for their products. However, legacy user interaction techniques have mostly originated in a world of desktop computers and more recently were adapted for smartphones. Therefore, their design concepts and technologies are mainly focused on supporting user interaction with digital systems. In contrast, SBEs bridge the digital and the physical worlds, thereby introducing a different interaction context.

Over the last few years, HCI techniques and technologies have evolved significantly (e.g. mixed-reality headsets, brain-computer interfaces). However, we still have a long way before we reach the point where the amazing rewards of those techniques and technologies reflect on SBEs. This way can be made shorter by simultaneously deploying and adapting existing techniques while developing and adopting new ones to make better use of recent advances in interaction technologies. The challenges described in section 1.1 and the realization of how crucial the support for user interactions is to the wide adoption of SBEs have motivated
us to propose a complete solution framework that aims at supporting user interactions in SBEs by providing multimodal UIs while minimizing the required interactions and mental workload to obtain and use those UIs. Before proceeding with the details of the proposed framework, let’s first answer a few motivating questions.

1.2.1 Why CPS?

Legacy building automation systems, intelligent buildings, and smart buildings were often closed systems that rely solely on their local resources to realize their capabilities. In contrast, CPS are open systems that are connected to the Internet. This can provide several rewards compared to closed systems as follows:

- **Remote interaction.** Besides supporting local user interaction, an SBE may allow its users to interact with it from a remote location. The user can monitor the SBE, take actions, and receive feedback over the Internet.

- **Cloud resources.** Unlike closed systems, an SBE is not limited to its local resources but rather can benefit from the various cloud resources and services that are available through the Internet. The capabilities of closed system are limited by the resources they can provide locally. In contrast, the cloud provides a scalable solution to provide almost unlimited resources. This allows for collecting large volumes of data and performing computation-intensive tasks that were not possible or feasible with closed systems.

- **Data sources.** Unlike closed system, an SBE is not limited to its local data sources (e.g. sensors, historical databases, etc) but rather can reach other data sources (e.g. social networks, online calendars, etc) through the Internet. Besides the locally collected data, an SBE can leverage the data it collects through the Internet to take better actions and enhance the user experience of the SBE.

- **Collaboration.** Several SBEs at different locations may communicate over the Internet and collaborate to provide their users with a seamless experience as they move...
from one SBE to another. Combining data from several SBEs can provide valuable information that a single SBE cannot provide.

Besides the rewards mentioned above, an SBE as a CPS imposes security and privacy concerns. Being connected to the Internet makes an SBE subject to cyber attacks. Those concerns should be addressed appropriately to ensure a secure SBE that protects the privacy and ensures the safety of its users.

1.2.2 Why Explicit Interaction?

Ideally, an SBE would completely take over the control and meet all the needs of its users through autonomous actions. Indeed, it has been shown that humans are willing to give up partial control and accept a large degree of autonomy as long as the reward in usefulness is greater than the cost of limited control [Barkhuus and Dey, 2003]. However, realizing such SBE requires the provision of powerful tools and mechanisms that can infer all the relevant knowledge, factors, and constraints that can directly or indirectly impact the user experience before taking the appropriate autonomous actions that are guaranteed to fulfill user needs while accommodating for the conflicting needs. In other words, this requires an SBE not only to keep track of its surroundings but also to be fully aware of all the rules and goals that apply besides having the ability to read people minds and predict their reactions to planned autonomous actions, which is not yet practical. Autonomous actions have the potential to fail to meet user needs due to the lack of optimal methods and/or the absence of complete knowledge upon taking the decision. Moreover, a user might perceive autonomous actions as patronizing. Therefore, supporting explicit user interaction in SBEs remains a must.

1.2.3 Why Multimodal Interaction?

Supporting explicit user interaction with things requires providing UIs for those things. Those UIs, for several reasons as mentioned in section 1.1, often take the form of smartphone
apps or web-based portals. In both cases, the user is provided with a graphical user interface (GUI) that is displayed on a 2-dimensional (2D) screen. Relying solely on GUIs to support user interaction with things can degrade the user interaction experience for the following three reasons.

- **Mental workload.** A GUI requires a user to develop a mental model that maps things with physical embodiments and located in a 3-dimensional (3D) space to an abstract layout presented on a 2D screen. This mapping can induce an increased mental workload, especially as the number of things increases. For instance, consider a room with a few light sources distributed in the 3D space and a GUI that maps those light sources to a list of GUI items displayed on a 2D screen from which a user may select an item to control the corresponding light source. In this example, it can be tricky to the user to correctly map each light source to its corresponding item. A GUI may facilitate the mapping by allowing a user to give each light source a convenient name. However, as the number of light sources increases (e.g. a room with tens of ceiling light sources), giving a convenient name to each light source can be a tricky task on its own not to mention memorizing those names for later use.

- **Limited interaction.** Besides requiring the user to develop a rather complex mental model of the SBE, a GUI has limited input support. A user often provides input by pressing a physical button or tapping a touchscreen. In both cases, the user is not leveraging the full capabilities of the human body to engage and interact with the SBE. From the SBE point of view, the user is a creature with a single finger that provides the input and a single eye that sees the 2D visual output. Consequently, relying solely on GUIs to support user interaction with SBEs can result in a significantly limited interaction experience.

- **Inefficient interaction.** Accomplishing different tasks in an SBE involves different interaction scenarios. A GUI may require the user to perform more interactions in order to accomplish a given task. For instance, turning a light source on/off using
a smartphone app may be more cumbersome compared to simply turning a physical switch on/off, especially if that switch is easily reachable. Consequently, relying solely on GUI to support user interaction with different things in an SBE may not always allow for feasible interaction.

More recently, smart speakers (e.g. Google Home, Amazon Echo) were introduced as devices that support voice-based interaction. They allow for hands-free interaction between users and things in an SBE, where a user may provide input using voice commands and receive the output via audio playback. Similar to the three drawbacks of GUIs mentioned above, voice-based UIs as well suffer from the excessive mental workload, limited interaction, and inefficient interaction as illustrated below.

- **Mental workload.** A voice-based UI requires a user to develop a mental model that maps things to pronounced names and their functionalities to voice commands. Smart speakers allow a user to associate a convenient name per thing. However, this mapping can induce an excessive mental workload, especially as the number of things increases. Moreover, memorizing and recalling numerous voice commands to trigger different actions can be cumbersome. Requiring the user to recall the UI rather than recognizing it can introduce a significant mental workload.

- **Limited interaction.** Voice-based interaction does not leverage the full capabilities of the human body to engage and interact with the SBE. A user can only provide input using voice commands and receives the output via audio feedback. From the SBE point of view, the user is a creature with a mouth and two ears. Moreover, smart speakers support a limited set of spoken languages. This places a language barrier between an SBE and a user who does not speak any of the supported languages.

- **Inefficient interaction.** A voice-based UI requires the user to listen carefully in order to capture the audio output. Otherwise, the user may fail to capture the audio output and need to repeat the command. Compared to a GUI, where a user may capture the
output at a glance, voice-based output requires the user to pay attention as long as the audio output is playing. Attention is a scarce resource and many of us already suffer from information overload. When a voice-based UI has output, it typically engages the user immediately. Interrupting users to provide output can deviate them from focusing on their current tasks, which can have a disruptive effect on both task performance and emotional state [Bailey et al., 2001].

Different UI types have varying pros and cons. Whether the user would prefer to use one UI type or another to accomplish a given task depends on the task at hand, the current context, and the user’s personal preference at the moment. Therefore, an SBE should provide its users with a variety of UI types allowing for a flexible interaction experience. Supporting multimodal interaction in SBEs is crucial to overcome the limitations associated with different UI types and to leverage the full sensing and actuation capabilities of the human body allowing for better interaction and engagement between users and SBEs.

1.2.4 Why Not Vendor-Provided Apps?

As mentioned in section 1.1, the current trend that is followed by most thing vendors is to provide UIs for their products in the form of smartphone apps and/or web-based portals. However, relying on custom vendor-provided apps to support user interaction with things can degrade the user interaction experience for the following two reasons.

- **Scalability.** As the number of SBEs and the number of things they incorporate continue to increase, the number of vendor-provided apps will increase as well. This will require a user to obtain, install, master, and use numerous apps. Moreover, interacting with different things will require the user to switch between different apps. Switching to the appropriate app to accomplish a given task requires the user to perform additional interactions, which introduces an overhead that can degrade user performance. Moreover, as the number of apps increases, selecting the correct app can be
error-prone. Furthermore, the lack of inter-usability between different apps provided by different vendors can confuse the user and introduce a significant mental workload. Consequently, relying on custom vendor-provided apps to support user interaction with things does not scale well.

- **Prior knowledge.** Non-domestic users of an SBE (e.g. visitors) often do not have prior knowledge about the SBE nor the things it incorporates. Therefore, it can be tricky for them to figure out, without guidance, what apps or web-based portals they need to use in order to interact with that SBE. Consequently, they might waste the chance to fully benefit from the capabilities of the SBE. Moreover, it might be infeasible for non-domestic users to install and master new apps, especially during short visits.

### 1.3 Methodology

This section gives a summary overview of our solution approach and describes how different components fit in parts of this dissertation document. Things and their UIs are often decoupled, where a thing UI is provided through a third party entity that has the required I/O capabilities. We refer to that entity as “Interaction Proxy” ([section 4.2](#)), which can take the form of a single device (e.g. a smartphone) or a set of devices that operate synergically to provide the UI. This way, the interaction proxy plays an intermediary role between a user and a thing in an SBE. Based on that, we describe a framework for explicit user interaction in SBEs ([chapter 5](#)). The framework defines how SBE users can interact with the incorporated things and the provided services. Clarifying the relationships between users, things, and interaction proxies in an SBE can help to understand the interaction environment, identifying its weaknesses, and determining potential improvements.

Studying the explicit user interaction framework revealed that relying on different UI implementations for different things and different SBEs can introduce several interaction complications to the users. First, those UIs are usually heterogeneous and lack inter-usability
due to the fact that they are provided by different vendors. Second, as the number of things in an SBE increases it becomes more challenging to the users to control them individually. This will require the user to learn numerous heterogeneous UIs and to understand the relationships between different things and services. Moreover, manually switching between several UIs to accomplish different tasks can be error-prone and time-consuming, which can degrade the user’s performance besides increasing the mental workload. Assisting users by automatically providing them with the appropriate UIs rather than requiring them to obtain the UIs manually can help to enhance their performance as well as reducing the required mental workload. In order to achieve that, we propose a framework to automatically provide a consistent user experience in an SBE based on the current context and the available I/O capabilities (chapter 5). The framework allows an interaction proxy to discover the available SBEs, learn about the incorporated things and provided services of the designated SBE, and provide the users with dynamically generated UIs based on the current context and the available I/O capabilities. In order to realize that framework, we define the following main research tasks.

- **Maintaining thing/service information.** An SBE should maintain information about the things it incorporates and the services it provides. This can be challenging due to the heterogeneity between different things. Several semantic approaches have been proposed in the literature to address interoperability problems. This research task focuses on providing an ontology-based approach to store thing/service information as well as providing a registration service that allows an SBE to register its things/services.

- **Discovering SBEs.** A user should be able to discover the available SBEs without having prior knowledge about their existence. The available SBEs may refer to SBEs in user surroundings or SBEs at a remote location. In either case, the interaction proxy should be able to discover them autonomously. This research task focuses on providing a technique that the framework can utilize to provide an SBE discovery service.

- **Learning about an SBE.** Once the interaction proxy discovers the available SBEs,
the SBE of interest can be selected either manually via explicit user input or automatically using context information. In either case, the interaction proxy needs to learn about the incorporated things and the provided services of the selected SBE before it can provide the corresponding UI. This research task focuses on exploring possible alternatives to generate a machine-readable SBE description that provides the required information to generate the UI while being generic enough to allow for generating different UI types for different interaction proxies with varying technologies and I/O capabilities.

- **Generating the UI for an SBE.** An interaction proxy should have the ability to automatically generate a UI for an SBE given a machine-readable description of that SBE. The UI should adhere to the I/O capabilities of the interaction proxy. Therefore, the UI types that can be provided may differ from one interaction proxy to another. This research task focuses on exploring different UI types and how to generate them from the provided SBE description.

- **Adapting an SBE UI.** Leveraging context information, the UI should adapt to the current context in order to facilitate user tasks and reduce the required interactions. This research task focuses on supporting autonomous UI navigation and functionality filtration based on context information.

We use an IoT testbed developed around an interdisciplinary research project, FutureHAUS, to implement the developed framework. The testbed supports creating and connecting various things with embedded sensors and actuators. It allows for collecting data, exploring patterns of use, and testing different alternatives to support user interaction. The FutureHAUS things include appliances, lights, furniture elements, floors, etc. They provide various presentation medium, from tablets to multi-touch displays and mixed-reality devices.

We use OSIsoft PI System (http://www.osisoft.com), a data collection and analysis platform, as a part of the framework implementation. It can handle time-series input signals
with different formats to store collected data and thing/service descriptions. Its RESTful API is used by the framework services. Communication between things and interaction proxies takes place using the Message Queue Telemetry Transport (MQTT) protocol, a light-weight publish-subscribe communication protocol. The conducted experiments/studies include determining the communication delays to ensure interactivity and supporting multi-modal interaction by providing different UI types, including GUI, voice-based, gesture-based, and mixed-reality based UIs.

1.4 Contribution

The contributions of this research can be summarized as follows:

- State-of-the-art literature review of the IoT covering its enabling technologies, application domains, and research directions.

- State-of-the-art literature review of SBEs covering terminologies and related concepts with focus on user interaction support.

- Characterization of the challenges associated with supporting user interaction in SBEs and the limitations of the current approaches.

- The “Interaction Proxy” concept as a core component of a proposed framework for explicit user interaction in SBEs.

- A framework to support user interaction with SBEs by providing consistent and multimodal UIs in a scalable fashion.

- A simulation tool for MQTT-based communication to evaluate the communication performance before the actual implementation.

- A gesture-based interaction technique that allows for controlling individual light sources by turning them on/off and adjusting their parameters such as brightness and color.
• Extending embodied interaction in MR environments allowing a user to interact with virtual objects using the whole body rather than using a limited set of gestures.

• Participating in building a testbed around the FutureHAUS project that allows for collecting usage data and exploring different interaction designs.

• An example user study that explores different modalities of interaction to control indoor lights.

1.5 Document Organization

Besides the introduction and conclusion chapters, this document consists of three main parts:

• **Background and Related Work.** This part consists of two chapters. In [chapter 2](#), we present a state-of-the-art literature review of the IoT including its enabling technologies, applications, challenges, and research directions. In [chapter 3](#), we focus on SBEs as one of the application domains of the IoT. We discuss their evolution from closed systems to IoT-based systems and their support for user interaction.

• **Problem and Approach.** This part consists of two chapters. In [chapter 4](#), we describe the problem scope and identify the challenges associated with supporting user interaction in SBEs before formulating the research objectives. In [chapter 5](#), we present a framework that describes explicit user interaction in SBEs. Furthermore, we propose a framework to provide UIs for SBEs.

• **Evaluation and Discussion.** This part consists of three chapters. In [chapter 6](#), we present a developed simulation tool that can simulate the network traffic in SBEs. In [chapter 7](#), we use the FutureHAUS as a testbed to explore different UI types and interaction techniques that can help users interact with an SBE. In [chapter 8](#), we describe a pilot user study that was conducted to compare the usability of different interaction modalities when used to control lights in SBEs.
Part I

Background and Related Work
Chapter 2

The Internet of Things

Since its inception, the Internet has mainly focused on supporting human-to-human interaction through interconnected devices, where humans are the dominant providers and consumers of Internet traffic. According to this vision, the digital world and the physical world are almost isolated from each other with humans acting as the main bridge between both worlds. Humans are responsible for collecting data about the physical world and feeding them into the digital world as well as taking actions to change the state of the physical world based on digital information.

The Internet of Things (IoT) extends this vision by enabling direct interaction between the digital and the physical worlds. In order to realize such interaction, the IoT incorporates smart devices, called things, into the Internet. Things can interact directly with the surrounding physical environment without human intervention. They can collect data about physical phenomena using a variety of sensors and can perform actions to change the state of the physical environment using actuators.

The term “Internet of Things” was introduced in 1999 as the title of a presentation made by Kevin Ashton at Procter and Gamble (P&G) in the context of supply chain management [Ashton, 2009]. In 2001, MIT Auto-ID center presented their IoT vision as an in-
telligent infrastructure linking objects, information and people through the computer network [Brock, 2001]. In 2005, the International Telecommunication Union (ITU) explored the IoT key technical vision, enabling technologies, market potentials and emerging challenges in the “Internet of Things” report, the seventh in the series of ITU Internet Reports [Strategy and Unit, 2005]. In 2008, the US National Intelligence Council (NIC) included IoT in a list of six disruptive civil technologies with potential impact on US interests [Council, 2008]. NIC foresees IoT as a potentially disruptive technology because of the risks that will arise when people can remotely locate, monitor, and control things as well as the information security risks caused by having Internet nodes residing in everyday things.

The Internet of Things (IoT) covers a wide range of technologies and research efforts. Many research communities have been involved in establishing technologies and solutions that support the realization of the IoT vision. Such communities include Internet of Things (IoT), Mobile Computing (MC), Pervasive Computing (PC), Wireless Sensor Networks (WSNs), and Cyber-Physical Systems (CPS) [Stankovic, 2014]. The diversity of the involved research societies and the variety of challenges that researchers need to address at different levels have led to a difference in viewpoints. This has caused some confusion about what the term “Internet of Things” really means. In [Atzori et al., 2010], the authors identified three main visions for the IoT paradigm: Internet-oriented, things-oriented, and semantic-oriented. The Internet-oriented vision focuses on networking issues and protocols. The things-oriented vision focuses on integrating things into a common framework. The semantic-oriented vision focuses on the representation, storage, and analysis of the exchanged information.

Several definitions for the IoT have been proposed in the literature. Among the Internet-oriented definitions for the IoT is the definition provided in the report “Internet of Things in 2020: A Roadmap for the Future” that outlined the results of the workshop “Beyond RFID The Internet of Things” [Bassi and Horn, 2008]. The report defined the IoT based on the semantic origin of the expression, which consists of two concepts: “Internet” and “Thing”. Borrowing from the definition of the term “Internet” as “the worldwide network of interconnected computer networks, based on a standard communication protocol, the
Internet suite (TCP/IP)” and the definition of the term “Thing” as “an object not precisely identifiable”, the report defines the term “Internet of Things” as “a worldwide network of interconnected objects uniquely addressable, based on standard communication protocol”.

Among the thing-oriented definitions is the definition provided in [Tan and Wang, 2010]. The authors introduced a functionality and identity-centric definition, where they defined the IoT as “things having identities and virtual personalities operating in SBEs using intelligent interfaces to connect and communicate within social, environmental, and user contexts”.

Another thing-oriented definition can be found in [Gubbi et al., 2013], where the authors introduced a user-centric definition that is not restricted to any standard communication protocol. They defined IoT for smart environments as “interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications”.

The Cluster of European Research Projects on the Internet of Things (CERP-IoT) defined the IoT as “a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols where physical and virtual things have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network” [Sundmaeker et al., 2010]. This definition for the IoT covers both the Internet-oriented and the thing-oriented perspectives. The CERP-IoT also defined things as “active participants in business, information and social processes where they are enabled to interact and communicate among themselves and with the environment by exchanging data and information sensed about the environment, while reacting autonomously to the real/physical world events and influencing it by running processes that trigger actions and create services with or without direct human intervention”.

This definition of “things” considers the semantic-oriented perspective as well. From a thing-oriented perspective, the authors in [Miorandi et al., 2012] defined a thing as an entity that has a physical embodiment, a unique identifier, a minimal set of communication functionalities, some basic computing capabilities, and the capability to sense physical phenomena and trigger actions having an effect on the physical reality.


2.1 IoT Enabling Technologies

The IoT consists of three main components, namely hardware, middleware, and presentation [Gubbi et al., 2013]. The hardware consists of sensors, actuators, and embedded communication devices. The middleware provides storage and data analytics tools. The presentation component uses interpretation and visualization tools to present the data to the user in an easy to understand manner. The IoT hardware will involve billions of interconnected things that need to communicate and sense the surrounding environment. Therefore, identification, communication, and sensing technologies are essential to enable the IoT. The middleware needs to handle a tremendous amount of collected data, which introduces a massive demand for storage and processing resources. The realization of the IoT vision is achievable through the integration of several enabling technologies. This section discusses these technologies and the role they can play in the IoT.

2.1.1 Wireless Communication

Wireless communication technologies provide mobility, convenience, flexibility, and lower cost of communication compared to wired communication. Consequently, wireless communication will play a key role in providing connectivity for the majority of IoT devices. The reduction in terms of size, energy consumption, and cost of wireless communication devices will allow their integration into almost all everyday objects. On the other hand, eavesdropping on wireless communications is extremely simple, which makes it more vulnerable to security attacks.

One important resource for wireless communication is the frequency spectrum. Given the limitation of the natural frequency spectrum and the increasing needs for higher data rates, spectrum scarcity can be a major challenge [Yüce and Arslan, 2009]. Cognitive radio [Mitola III and Maguire Jr, 1999] allows for better utilization of the radio frequency spectrum, where the transceiver can sense its environment to detect which communication...
channels are in use and which are not, and then it can use the unoccupied channels for transmitting data. Since both cognitive radio and IoT technologies are still in their infancy, forming a coalition of both technologies requires addressing many challenges such as interoperability of interconnected devices, adaptability to the surrounding environment, and standardization [Shah et al., 2013].

2.1.2 Radio Frequency Identification

Radio frequency identification (RFID) is a wireless communication technology that allows for uniquely identifying tagged objects or people. In the IoT, identification of things is essential and hence RFID can play a key role as an identification technology. An RFID system consists of three main components: a tag, a reader, and a host [Hunt et al., 2007]. A tag, or transponder, consists of a semiconductor chip, an antenna, and sometimes a battery. Acting as electronic barcodes, tags have unique identifiers, which makes tagged-objects identifiable. A reader, or transceiver, consists of an antenna, a radio frequency electronic module, and a control electronic module. Using radio waves, a reader can communicate with many tags simultaneously to read data from or write data to them. A host, or controller, is usually a PC or a workstation running a control software and a database that has information records associated with different tags. The reader can scan for tags in the surrounding environment to detect the presence of tagged-objects. When a tag enters the reading zone of a reader, the reader sends a signal to the tag and the tag responds with its stored data. Afterward, the reader communicates the data received from the tag to the host for further processing via a standard network interface.

There are many types of RFID tags with varying capabilities [Koschan et al., 2006]. Tags can be chip-less or chip-based and chip-based tags can be read-only, write once/read many, or read-write. In terms of powering, there are three main types of RFID tags: active, passive, and semi-passive tags. Active tags are battery powered and can transmit a signal to the reader with up to 100 ft. Passive tags have no internal power source and they use power
harvesting to derive the power required to respond to the reader signals from the readers electromagnetic field, which generates a current into the tag antenna by induction. Semi-active tags use batteries to power the microchip while receiving the signal from the reader and use energy harvesting to power the transmission of tag response to the reader. Active tags have higher transmission range compared to passive tags but passive tags are much less expensive.

RFID is rapidly becoming a cost-effective technology and is expected to replace the traditional barcodes. Although barcode systems are much less expensive, RFID systems can provide many benefits that barcodes systems cannot provide including: the ability to both read and write to tags, the ability to function without direct line of sight between tag and reader, the ability to use encryption for data security, and the ability to store more information using larger memory size [Hunt et al., 2007]. Nowadays, RFID devices and solutions are in use in many application areas, in particular in the goods supply chain management and logistics sectors [Miorandi et al., 2012]. RFID systems can identify and track surrounding objects in real-time, which supports the IoT vision by providing a mean for mapping the physical world into the virtual world [Atzori et al., 2010].

### 2.1.3 Near Field Communication

Near Field Communication (NFC) is a short-range radio communication technology that enables a device to communicate with another by bringing them into proximity at a maximum distance of around 10cm or less [Want, 2011]. A user can interact with a smart object that supports NFC, such as NFC tag, using an NFC-enabled mobile phone by simply touching them to each other.

As more phone manufacturers include NFC chips in their mobiles, this will encourage the development of a variety of services such as advertising, payment and loyalty applications, and access control systems. Advertising may take the form of transmitting a Uniform Resource Locator (URL) to customers smartphone so that the customer can navigate to a website
that contains further information about the advertised product or service. NFC allows for integrating loyalty and credit cards into mobile phones, which makes payments more convenient with the use of a single physical instrument. Moreover, an NFC device can act as an access key that replaces key fobs and cards currently in use with access control systems. The simplest use case of NFC is where two users communicate and exchange data by touching their mobile phones to each other.

NFC communication occurs between two compatible devices with an operating frequency of 13.56 MHz and a data transfer rate of 106, 212, 424, or 848 kbps [Curran et al., 2012]. NFC supports two communication modes: active and passive. In the active communication mode, communication occurs between two active NFC devices that can generate radio frequency field. In the passive communication mode, only one device, called initiator, is active while the other device, called target, is passive. The initiator device can generate radio frequency field to initiate the communication and control the data exchanges. The target device responds to requests from the initiator using load modulation [Nagashree et al., 2014]. In the active communication mode, only two devices can communicate at a time whereas in passive communication mode the initiator can communicate with several targets at a time.

The NFC technology utilizes three NFC devices, namely NFC mobile, NFC tag, and NFC reader and supports three operating modes, which are reader/writer, peer-to-peer, and card emulator [Coskun et al., 2013]. In reader/writer mode, communication occurs between an NFC mobile and an NFC tag using NFC Data Exchange Format (NDEF) to exchange data messages. In peer-to-peer mode, communication occurs between two NFC mobile devices. Finally, in card emulator mode, secured data transfer occurs between an NFC mobile and an NFC reader where the NFC mobile acts as a standard smart card.

The NFC technology is here to stay given the wide deployment by smartphones manufacturers and the variety of applications that can provide a more convenient way to accomplish everyday tasks. Moreover, NFC is backward compatible with RFID [Curran et al., 2012]. This gives NFC enabled mobile phones the capability to read RFID tags and interact with
traditional RFID systems. IoT applications can benefit from NFC enabled devices carried by millions of users around the world by leveraging identification and tracking information that the NFC devices can provide.

### 2.1.4 Wireless Sensor Networks

Advances in wireless communication and electronics have enabled the development of low cost, low power, and small size sensor nodes. Sensor nodes are devices with limited power and computing resources incorporating one sensor or more, a processor, memory, a power supply, a radio, and sometimes actuators depending on the application and the type of sensors used [Yick et al., 2008]. Sensor nodes are usually battery-powered and may use energy harvesting through solar cells or scavenge energy from motion or wind depending on the application and the deployment environment [Stankovic, 2008]. A variety of sensors can be attached to a sensor node including thermal, biological, chemical, optical, and magnetic sensors. A wireless sensor Network (WSN) consists of a large number of sensor nodes, ranging from hundreds to thousands, that are densely deployed either inside the phenomenon of interest or very close to it. The deployment topology does not need to be engineered or predetermined, which requires a self-organizing capability to allow for a random deployment in inaccessible terrains, disaster relief operations, or hostile environments [Akyildiz et al., 2002].

The sensor nodes monitor physical or environmental conditions and cooperatively transmit sensory readings using wireless communication in a single or multi-hop fashion to a fewer set of special node called sinks, which can use the collected data locally or send it to another network through a gateway [Buratti et al., 2009].

Sensor nodes have limited lifetime due to power constraints and node failures are common for several reasons. Additionally, WSNs are deployed in an ad-hoc manner and sometimes incrementally, that is adding more nodes after the first deployment. Moreover, nodes may change their locations if they are equipped with actuators that allow for mobility. Therefore, the routing protocol must accommodate to such issues with and the network
should have a self-healing capability to ensure scalability and longevity of the deployed network [Prathap et al., 2012].

The IoT can benefit from the integration of both the RFID and the WSN technologies to support the identification and the sensing capabilities. RFID sensor networks (RSNs) extend RFID to include sensing capabilities while keeping the advantage of small, inexpensive, and RFID tags that can harvest energy from readers signal [Buettner et al., 2008]. In [Yang et al., 2011], the authors proposed a hybrid framework that integrates sensors, passive and active RFID systems into a unified architecture that can improve the overall performance of the humanitarian supply chain while reducing the cost, the complexity, and the time requirements compared to the use of individual systems.

2.1.5 Mobile Phone Sensing

Smartphones and other wearable devices can play a crucial role in the IoT. The number of smartphone users will surpass two billion by 2016, which makes them nearly ubiquitous. The embedded sensors together with the powerful computing and communication capabilities allow smartphones to sense, process, and communicate data about users and their surrounding environment. GPS can provide context information by combining collected data with location information. Moreover, smartphones can perform autonomous actions on behalf of the user. For example, a smartphone may call an emergency number based on a decision made by a health monitoring system. The wide deployment of smartphones provides a powerful infrastructure that different sensing paradigms can benefit from.

Participatory Sensing [Burke et al., 2006] aims to leverage the widespread of phones with rich sensing capabilities to form interactive sensor networks. Compared to the traditional sensor networks, participatory sensing involves the users in the sensing process, where they can collaborate in data collection using sensors embedded in their phones. The participants are fully aware of their role in the sensing process and they control what, when, how, and where to sample. Consequently, the quality of collected data depends on the fidelity of the
participants and the compatibility between their decisions at different stages of the sensing process and the goals of the application [Lane et al., 2010]. A participatory system should focus on addressing privacy concerns as well as providing the sharing tools and the social mechanisms in order to encourage the participation of phone owners.

On the other hand, opportunistic sensing [Campbell et al., 2006] shifts the burden from the phone owners to the sensing system, where the system automatically detects the state of the phones and utilize their sensors to collect samples when appropriate. Phone owners are no longer active participants in the sensing process and sampling occurs automatically when the state of a phone matches the application needs. A major concern is to protect user privacy during the collection of sensory data. Moreover, the collection of sensory data must be transparent, that is it has no significant effect on the normal user experience [Lane et al., 2008].

Compared to participatory sensing, opportunistic sensing has the potential to scale up and involve a large number of users regardless of their direct interest in the application. However, opportunistic sensing must provide mechanisms that address the privacy and transparency challenges while allowing for the collection of meaningful data. Moreover, while the participatory sensing paradigm can benefit from the intelligence of the participants to solve context problems (e.g. changing the state of the phone for the benefit of the application), the opportunistic sensing paradigm needs to provide automatic mechanisms to solve such problems (e.g. monitoring the phone state and collecting samples when appropriate) [Lane et al., 2010].

2.1.6 Cloud Computing

Cloud computing is a computing technology that provides transparent access to a dynamically scalable and often virtualized resources as a set of services over the Internet [Furht and Escalante, 2010]. According to the National Institute of Standards and Technologies (NIST) [Mell and Grance, 2011], “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources
(e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction”.

In the IoT, billions of interconnected devices are continuously collecting data about their surrounding environment and produce a massive amount of data streams. Storing, processing, and analyzing such huge data is a major challenge that cannot be handled by a single entity. Cloud computing technology offers a promising solution as it allows for a flexible and a scalable resource provisioning that can meet the tremendous storage and processing requirements of the IoT. Moreover, the cloud-computing model can lower the operating cost since the allocation and deallocation of resources occur on demand and there is no need for up-front investment.

Cloud computing can support IoT at different levels through the several service models it provides including software as a service (SaS), platform as a service (PaS), and Infrastructure as a service (IaS). Analytics and visualization tools can leverage the cloud computing technology to support massive data processing and provide the user with accessible web-based interfaces [Gubbi et al., 2013]. A major concept in cloud computing is resource virtualization which hides the heterogeneity of resources and allows for a seamless integration between them. Cloud computing concepts can be adapted for the IoT context to help to address the device heterogeneity problem. This can be a starting point to structure the IoT as a utility that can provide things as a service and allows for accessing things from different environments [Mingozzi et al., 2013].

There are several cloud-based services provided by different vendors for supporting IoT applications such as ThingSpeak, Nimbits, Open.sen.se, Kaa, Azure IoT Suite, Intel IoT Analytics, IBM Watson IoT, Oracle IoT Cloud Service, Cisco IoT Cloud Connect, Amazon Web Services IoT, and more. In the following subsections, we provide an overview of some cloud-based IoT services showing their varying features and supported functionalities.
2.1.7 Data Visualization

Although the IoT introduces the concept of direct interaction between the digital and the physical worlds without human intervention, this does not mean that IoT systems will operate autonomously and isolated from users. In fact, the IoT vision incorporates several interaction models including human-to-human, thing-to-thing, thing-to-human, and thing-to-environment interaction. Thus, there is need to support user interaction through convenient interfaces.

Given the tremendous amount of data generated continuously by IoT systems, it is infeasible for users to extract useful information by simply skimming through raw data. Visualization techniques can create a graphical presentation of information, which is much easier to understand by the users and can support knowledge inference and decision-making. Humans can acquire more information through vision than through any other sense and hence data visualization, if presented well, allows for a rapid interpretation of a sheer quantity of information [Ware, 2012]. A visualization can summarize large data sets, can help identifying trends or phenomena in data streams, and allows for user interaction in a variety of ways such as browsing, sampling, and specifying data fields to be displayed [Fayyad et al., 2002]. The advances in display technologies such as touch screens and 3D displays provide a good opportunity to develop new visualization techniques that use those technologies to provide a more attractive, a less complicated, and a meaningful visualization of data [Gubbi et al., 2013].

2.2 IoT Applications

Several application domains can benefit from the IoT including smart cities, smart environment, smart water, smart metering, security and emergencies, retail, logistics, industrial control, smart agriculture, smart animal farming, smart buildings, and health care [Asin and Gascon, 2012]. In this section, we illustrate those application domains and how they can benefit from the IoT.
2.2.1 Smart Cities

The main goal of smart cities is to use digital technologies to achieve a better use of the public resources and increase the quality of services offered to citizens while reducing the operational costs of the city administration [Zanella et al., 2014]. The IoT can contribute to several applications that can realize the smart city vision. A structural health monitoring application can monitor the structural health of city buildings, bridges, and historical monuments via appropriate sensors such as vibration and deformation sensors to monitor the building stress and temperature and humidity sensors to monitor the environmental conditions that may affect the structural health of buildings [Lynch and Loh, 2006]. Although the installation of sensors in buildings and interconnecting them into a monitoring system requires a considerable initial investment, the long run cost can be feasible compared to the cost needed for periodic structural testing by human operators. Moreover, the sensors can collect structural health data much more frequently and hence allows for proactive maintenance actions. A waste management application can monitor the level of load in waste containers via sensors installed in those containers. This allows for the optimization of collector truck route and hence reduce the cost of waste collection [Nuortio et al., 2006]. A noise monitoring application can monitor the level of noise using sound detectors or microphones installed in different city zones and build a space-time map of noise pollution [Maisonneuve et al., 2009]. However, the installation of microphones in public zones will probably raise privacy concerns. A traffic monitoring application can identify congestions based on information collected via several sensors including sensors and GPS devices installed in vehicles as well as noise sensors and air pollution sensors [Li et al., 2009]. Those sensors can provide a denser source of information compared to the traditional camera-based traffic monitoring systems. Moreover, a smart road may display warning messages according to the weather condition or in response to unexpected events such as accidents. An energy consumption application can monitor the energy consumption for the whole city and provide a detailed view of the amount of energy consumed by the different services, which allows for energy optimization. A smart parking application can monitor the park spaces available in the city and direct motorists
to parking slots [Lee et al., 2008]. Using RFID and NFC can allow for the verification of parking permits in slots reserved for residents or disabled. A smart lighting application can optimize the intensity of streetlights according to the time of the day, the weather condition, and the presence of people.

### 2.2.2 Environmental Monitoring

IoT technologies can support environmental monitoring applications and play a key role in sensing natural phenomena (e.g. temperature, humidity, wind, etc.) through distributed sensing devices that cover the area of interest. Processing the collected sensory data allows for the detection of anomalies that may represent a danger to human or animal life. IoT environmental monitoring applications can facilitate the monitoring of critical areas, whereby the presence of human operators might not be feasible or even applicable (e.g. remote areas, volcanic areas).

A fire detection application can use a variety of sensors, such as temperature and humidity, sensors to monitor preemptive fire conditions in forests and detect fire events more promptly than detection systems that relay of satellite imagery [Yu et al., 2005]. An air pollution application can monitor pollution emitted by factories and cars as well as toxic gases generated in farms and provide a map that displays air pollution levels and the corresponding locations in large metropolitan areas [Al-Ali et al., 2010]. An earthquake early detection application can use sensors deployed in specific places of tremors to provide early warnings for earthquakes [Youssef et al., 2007]. Moreover, the localization supported by the IoT through several technologies, such as GPS, RFID, and NFC, can help quickly locating survivors when an earthquake happens [Wang et al., 2011].

Monitoring of soil moisture, vibration, and earth density allows for the detection of dangerous patterns in land conditions. A snow avalanches detection application can use infrasonic sensors to monitor avalanche activity in a given area in order to compare predictions to actual events and assess the effectiveness of avalanche control by explosions [Thüring et al., 2015].
A variety of IoT applications can help monitoring of water level variations in rivers, detect chemical leakage and wastes of factories in rivers, and monitor pollution level and wastes in the sea.

### 2.2.3 Security and Emergencies

Security surveillance has become a necessity for several buildings and facilities. However, the widespread deployment of camera places a major concern about the privacy of individuals. Unlike other systems where the user can control the privacy level and decide what information to expose, an individual has no control over surveillance systems. The IoT can enhance the performance of current surveillance solutions while preserving privacy by collecting anonymous information. Most surveillance applications do not require the association of collected information with specific individuals except for specific rare situations. Thus, a monitoring system should keep the collected information anonymous and reveal the identity of the corresponding individuals only when necessary.

Security and surveillance systems can benefit from IoT technologies. Sensors monitoring the behavior of people can detect suspicious acts and allow for building systems that can trigger early warnings. Access control systems can control access to restricted areas and detect people in non-authorized areas. IoT technologies can provide a high level of flexibility where technologies such as FRID or NFC can identify individuals and allows for controlling their access to restricted areas.

IoT technologies can help maintain the safety of industrial facilities. A liquid presence application can detect liquid in data centers or warehouses and trigger an alarm to prevent breakdowns. A radiation monitoring application can monitor radiation levels in nuclear power stations surroundings and trigger leakage alerts. Similarly, sensors can detect gas levels and leakages in chemical factories and inside mines.
2.2.4 Retail and Logistics

RFID technology is already in use for inventory management and supply chain control. The RFID tags applied to goods act as electronic product codes (EPC) that identify products and allows for tracking them throughout the supply and delivery chain. RFID systems can benefit from the IoT by integrating with other systems rather than being closed system. However, this requires standardization effort to build RFID systems that can integrate seamlessly with other IoT systems. Intelligent shopping applications can provide advice at the point of sale according to customer preferences, health condition, or expiration dates of the products. NFC technology can support payment processing at the point of sale. Moreover, NFC can allow for payment based on activity duration for public facilities such as transport, parking, gyms, etc. RFID and NFC technologies can realize real-time monitoring of products at every link in the supply chain and supports after-sales services. The ability to obtain product information instantly and accurately enables enterprises to respond to market changes in the shortest time and enables customers to know about the availability of products and get more information about them. IoT technologies allow for monitoring the quality of shipment conditions through a variety of sensors that can monitor vibrations, temperature, humidity, and other environmental parameters. Locating items in warehouses and incompatible storage detection are possible with appropriate IoT applications.

2.2.5 Smart Buildings

A smart environment can enhance the user experience by providing easy and flexible ways to interact with the surrounding environment (e.g. home, office, etc.) as well as allowing for performing automatic actions based on decisions inferred from collected data and user preferences. Integrating IoT technologies in buildings can help in both reducing the consumption of resources associated with the building as well as in improving the experience of individuals populating those buildings. A power consumption monitoring system can provide the user with advice that can help reduce the consumption level or take actions on
behave of the user to save energy such as turning off specific appliances. A personal comfort system can maintain a level of room temperature and lighting that matches user preferences. A recommendation system may recommend TV programs to specific users based on information extracted from their social network profiles. Incorporating IoT technologies in buildings opens the gates to an unlimited number of applications that can provide a new way of living by taking smart actions based on information collected about both the building and the inhabitants. Several IoT systems that serve different purposes may coexist in the same building. Integrating those systems may allow for smarter decisions based on fusing the information provided by each of those systems. However, such integration requires standardization as well as addressing any conflicts that might arise. For example, the power consumption monitoring system may decide to reduce the intensity of lights to save energy while the personal comfort system may decide to keep the lights at high intensity. Several applications will require accurate data association of the data collected about inhabitants to be able to provide personalized decisions. Data association can be straightforward if the inhabitants are wearing or holding identifiable devices such as RFID tags. However, probably this will not be the case and other techniques for identifying inhabitants such as face or voice recognition may be useful.

2.2.6 Healthcare

The healthcare sector can benefit from IoT technologies in a variety of ways. In assisted living solutions, patients can use wearable sensors to monitor parameters such as body temperature, Heartbeat, and blood pressure. The sensors will periodically collect information about health parameters and send it to remote medical centers. This allows for remote monitoring of patients and providing rapid response actions when needed. A fall detection system can assist elderly or disabled individuals living independently by calling for help when needed. Identification technologies such as NFC can help identifying a patient upon entering a hospital and save the time needed for paperwork. Moreover, such automatic identification
will allow for faster access to the medical records of the identified patient, which might be crucial especially if the patient condition is critical. Additionally, people can track their daily life activities using appropriate sensors and IoT applications that can provide them with personalized suggestions for enhancing their lifestyle and avoiding health problems.

## 2.3 IoT Research Directions

The realization of the IoT vision requires efforts from several fields and research groups. This section illustrates the different research directions related to the IoT and identifies a set of research challenges in each direction.

### 2.3.1 Networking and Communication

Interconnecting a set of devices requires the ability to address those devices uniquely. IPv4, which is currently in use, uses only 4 bytes to address each device. The number of available IPv4 addresses is decreasing rapidly due to the continuous growth in the number of interconnected devices. The IoT will involve billions of interconnected devices and the IPv4 addressing scheme will not be able to handle such addressing demand. IPv6 uses 6 bytes to address each device and hence it provides a reasonable solution to the current scarcity of IP addresses in the IPv4 scheme. IPv6 allows for addressing approximately $3.4 \times 10^{38}$ devices, which should be enough to address any device or thing in the IoT. However, this requires all things to support the IPv6 scheme, which can be problematic for some already established technologies and deployed systems involving legacy devices.

RFID tags, which are mainly used in supply-chain management systems, use either 64 or 96-bit identifiers to represent Electronic Product Code (EPC) and have no native support for IP. Several solutions have been proposed to enable the integration of RFID identifiers and IPv6 addresses. In [Lee et al., 2007](#), the authors proposed a mapping mechanism for converting
the EPC identifier used in 64-bit tags into an IPv6 address by concatenating the 64-bit network prefix and the 64-bit EPC producing a 128-bit IPv6 address. In [Yoon et al., 2008], the authors introduced a network mechanism that uses an address management agent. The agent receives RFID tag identifiers of any length, generates corresponding IP addresses, and stores the mapping information to allow networking between RFID tags using IP.

The IPv6 over Low Power Area Networks (6LoWPAN) protocol [Mulligan, 2007] enables the exchange of IPv6 packets over low power wireless networks and hence allows for realizing the IoT vision by incorporating things with limited resources and capabilities into the Internet. However, IPv6 and 6LoWPAN are not widely supported by sensors and devices nor many of the currently deployed systems. The Global-IP protocol [Jara et al., 2012] provides a mechanism that allows for IP connectivity between devices employing technologies that have no native support for IP, such as Bluetooth Low Energy (BT-LE).

Usually, users do not use IP addresses directly to access resources over the Internet but rather they use Uniform Resource Locators (URLs), which are unique human-readable names associated with different resources. Domain Name Servers (DNS) can map the queried URL to the corresponding IP address. Similarly, in the IoT, there is a need for Object Name Servers (ONS) that can map human-readable object names into IP addresses. EPCglobal defined a DNS-based object naming service that uses the EPC stored in RFID tags to retrieve information about products from the EPC network [Fabian and Gunther, 2007].

Due to the massive number of devices collecting data on a continual basis, the IoT will involve unprecedented amounts of data traffic that may overwhelm the network and cause communication issues. Therefore, advanced network technologies and new standards may be needed to address this challenge. Traffic characterization is essential to support quality of service (QoS) in IoT.

Given the heterogeneity of the devices and the diversity of applications, the IoT will involve multiple traffic types with extremely different characteristics [Atzori et al., 2010]. Different traffic types usually require different QoS deliveries [El-Sayed et al., 2008]. Real-time appli-
cations are delay sensitive, while the delay-tolerant applications are mainly concerned with throughput. Meeting the QoS requirements in the IoT is a big challenge given the scale and the diversity of IoT applications.

2.3.2 Architecture

The IoT will involve heterogeneous devices provided by different vendors. Those devices will have variations in their capabilities and may use different standards. Therefore, there is a need for an architecture that can integrate such a variety of devices and allow cooperation between them. Object virtualization can provide a standard interface that exposes the features and functionalities of objects regardless of their technical implementation details. Previous systems involving sensors and actuators were mostly closed systems operated by a single organization to serve a predefined specific goal. Thus, applications for such systems were relatively simple and direct interaction between applications and devices was common. On the other hand, the large scale of IoT systems raises the need for a middleware that can provide the interoperability between different devices and applications. A middleware acts as an intermediary that abstracts and standardizes the interaction between things themselves and provides a unified interface for applications and users that hides the specific peculiarities of those things. Several Architectures for the IoT have been proposed and the majority of them follow a Service Oriented Architecture (SOA) approach.

In [Atzori et al., 2010], the authors defined a middleware of three layers: object abstraction, service management, and service composition, where the management of the trust, privacy, and security is the responsibility of all layers. The object abstraction layer exposes device functionalities through a standard web service interface and translates messages into device-specific communication commands. The service management layer contains a service repository of services associated with each object. The service composition layer allows for creating complex services by composing services provided by the service management layer and allows the application to interact with objects through the provided services.
In [Domingo, 2012], the author proposed an IoT architecture composed of three layers: perception, network, and applications layer. The main function of the perception layer is to identify objects and gather information. The network layer is responsible for transmitting information obtained from the perception layer. The application layer consists of a set of intelligent solution that meets user needs.

In [Jia et al., 2012], the authors divided the IoT system architecture into three layers: the perception layer, the network (or transport) layer, and the service (or application) layer. The perception layer is concerned with perceiving and collecting information about the physical world. The network layer provides transparent data transmission capability as well as an efficient, reliable, trusted network infrastructure platform to the upper layer. The service layer includes two sublayers: data management and application service. The data management sublayer is concerned with processing complex data as well as providing directory service and Quality of Service (QoS). The application service sublayer transforms information to content presented to applications and end users.

In [Da Xu et al., 2014], the authors proposed an IoT architecture composed of four layers: the sensing layer, the networking layer, the service layer, and the interface layer. The sensing layer integrates with hardware to sense and control the physical environment. The networking layer provides basic network support and data transfer over wireless and wired networks. The service layer creates and manages services. The interface layer provides interaction methods to the users and the applications.

Designing an architecture for the IoT is a big challenge that requires supporting many features including interoperability, openness, scalability, and robustness. The IoT will incorporate billions of heterogeneous devices connected through various types of networks with a variety of communication technologies and protocols. Therefore, the IoT architecture should support interoperability between the underlying technologies and provide users and applications with transparent services. In order to support openness, the architecture needs to provide unified interfaces and a common language to exchange information across different
IoT systems [Stankovic, 2014]. As the number of interconnected things increases scalability issues may arise, which may cause performance degradation. Supporting such a large scale of things in terms of naming, communication, device and service discovery, access authentication, protection, and control is essential for a successful IoT architecture. The openness and scale of the IoT imply high dynamicity, where things get connected on a continual basis while others may get disconnected probably due to failure. The architecture can support fault tolerance by replicating critical services and providing failure detection and recovery mechanisms. Over time, the IoT systems may suffer from deterioration. Things may become out of synchronization due to clock drifts, which may result in application failure. Location information may become inaccurate due to unexpected movement of things, which may result in incorrect inferences. Performance of actuators may degrade due to physical wear and tear, which may result in severe safety problems in some applications. In order to maintain a robust system, the architecture must provide self-healing mechanisms such as resynchronization of clocks, relocation of devices, and recalibration of sensors and actuators [Stankovic, 2014].

Integrating different IoT systems with each other and with existing legacy systems is another challenge. Although the sharing of IoT resources can significantly reduce the deployment cost, different systems may have conflicting needs and/or goals [Stankovic, 2014]. Solving such conflicts is crucial for a seamless integration between systems. The IoT architecture should carefully define the policies for system integration and resource sharing in order to ensure the correct operation of interacting IoT systems.

2.3.3 Web-of-Things

The web-of-things (WoT) term refers to the use of Web standards, protocols, and technologies to interconnect physical objects and integrate them into the World Wide Web. The way that the WoT relates to the IoT is similar to how the Web relates to the Internet. In the IoT vision, the physical world will integrate with the Internet, such that everyday
things become accessible in the digital world. Several competing standards and solutions have been proposed to achieve such integration. However, no solution has been widely accepted and more standardization effort need to take place. Alternatively, the authors in [Guinard and Trifa, 2009] proposed leveraging the success of Web 2.0 mashup applications for integrating physical things to the Web. The authors described two methods for integrating things to the Web using Representational state transfer (REST): direct integration of web-enabled devices and indirect integration through a gateway for resource-limited devices. The use of the REST principles allows for a straightforward and flexible integration of things into the Web, which can simplify the creation of applications for the IoT.

Realizing the WoT vision requires embedding web servers in devices with limited hardware capabilities. Fortunately, those embedded Web servers need to handle much less simultaneous connections compared to the traditional Internet Web servers. Moreover, Asynchronous JavaScript and XML (AJAX) techniques allow for transferring some of the workloads from the resource-limited servers to the more powerful clients [Guinard et al., 2011]. In [Duquennoy et al., 2009], the authors analyzed the Web protocols in terms of traffics and memory needs and based on this analysis they proposed an efficient event-driven architecture for embedded web servers.

### 2.3.4 Data Management and Knowledge Creation

The massive number of IoT devices implies a tremendous amount of generated data. Sensors will collect data about the physical environment on a regular basis producing continuous flows of data-streams. This introduces a number of challenges in terms of networking, storage, and processing. The networking infrastructure needs to support the enormous traffic that will flow between the interconnected nodes. Compared to the traditional Internet, the size of stored data in the IoT will continue to grow dramatically with much higher rates. Storage of collected data is necessary to allow for historical queries. Thus, storage and retrieval mechanisms are required to handle such massive amount of data in a scalable manner.
The key power of the IoT is its ability to collect a significant amount of data from several distributed sources. Raw data may not have much meaningful value and hence it cannot provide many benefits to users and applications. Therefore, methods to clean, manage, query, and analyze the collected data are needed to allow for knowledge creation and decision making [Aggarwal et al., 2013].

2.3.5 Security

Dealing with security attacks is a fundamental problem in computer systems. Securing IoT systems involves many challenges due to several reasons. First, things are physically accessible since they tend to exist everywhere and are usually unattended. Second, most things will communicate wirelessly and probably using less secure wireless media, which makes eavesdropping very simple. Third, random failures are commonplace and attackers can exploit them. Forth, most things have limited resources while the majority of security solutions involves heavyweight computations and have large requirements. Fifth, legacy devices may use obsolete security mechanisms or even lack them, which makes integration with those devices very difficult. Finally, IoT systems will exchange a massive amount of heterogeneous data, which requires efficient and scalable security solutions.

Securing IoT systems is crucial, especially for systems that incorporate actuators. Malfunctioning actuators can cause physical damage or harm people. IoT systems must be tolerant to security attacks that is being able to continue working in the presence of attacks and to recover from those attacks. A security solution should provide the system with self-healing mechanisms that are able to detect attacks, diagnose them, and apply repairs in real-time [Stankovic, 2014]. Given that a system is as weak as its weakest point, securing the IoT requires applying security mechanism at different layers of the IoT architecture and ensuring that those mechanisms complement each other to create an overall security framework. In order to achieve that, it is necessary to understand the characteristics and the security requirements of each layer in the IoT architecture.
Several IoT architectures have been proposed, as discussed in section subsection 2.3.2. Most proposed architecture focus on three main layers: perception layer, network layer, and application layer. The perception layer uses sensor nodes to collect information about its surrounding environment. The security requirements for the perception layer include node authentication as well as data integrity, authenticity, and confidentiality [Suo et al., 2012]. Node authentication is necessary to prevent illegal node access. Data encryption is necessary to protect the confidentiality of information exchanged between sensor nodes. Sensors have limited resources in terms of processing power and storage capacity. Therefore, heavyweight security mechanism, such as public-key encryption, are impractical to use in this layer. Thus, there is a need for lightweight encryption algorithms and key management techniques. The IoT can benefit from security solutions introduced for WSNs to achieve security at the perception layer in a resource efficient manner.

Asymmetric key encryption algorithms, such as RSA, involves much more computations compared to symmetric key encryption algorithms, such as AES. Moreover, a key management algorithm based on asymmetric encryption requires the exchange of fewer bits compared to asymmetric encryption. Thus, the energy required to perform computations and communications are significantly larger for asymmetric encryption compared to symmetric encryption [Carman et al., 2000]. Therefore, symmetric encryption seems to be a reasonable choice for securing communications and ensuring the confidentiality of data exchanged between the resource-limited sensor nodes. In [Lee et al., 2014], the authors proposed a lightweight encryption method based on XOR manipulation for enhancing the security of RFID systems.

Due to the limited communications range of sensor nodes and the ad-hoc deployment of sensor networks, the distribution of keys to sensor nodes relays on key pre-distribution. A sensor network may use a single mission key or a set of different keys, each shared between exactly two nodes. The problem with the use of a single mission-key, where all of the nodes use the same key, is that the capture of any node may compromise the entire network. On the other hand, the use of different keys requires each node to store a key for each other node, which is unnecessary because a node needs to communicate only with neighbors in its
communication range. Moreover, storing a key for each other node may be impractical for large networks involving thousands of nodes given the memory limits of sensor nodes. In [Eschenauer and Gligor, 2002], the authors introduced a key-management scheme for sensor networks that rely on probabilistic key sharing among the nodes of a random graph. Their schemes avoid the problems associated with the use of a single mission-key and require each node to store only a limited number of keys rather than storing a key for each other node. The key distribution consists of three phases: key-predistribution, shared-key discovery, and path key-establishment. The key-predistribution phase involves the generation of a large pool of keys and the installation of a small number of keys, drawn from that pool, on each sensor node such that any two nodes share at least one key with a chosen probability. In the shared-key discovery phase, each node discovers its neighbors with which it shares keys. The path-key establishment phase allows for the creation of a communication path between two nodes that do not share a key through one or more intermediate nodes. Consequently, with much fewer keys stored at each node, a node can communicate with any other node either directly if they share a key or indirectly via a communication path between them through other nodes.

The network layer is responsible for transmitting information obtained from the perception layer. It should provide an efficient, reliable, trusted network infrastructure platform to the upper application layer. The security mechanisms used in the Internet, which are relatively mature, can help secure the IoT network layer. However, those mechanisms need further investigation to prove their scalability and adaptability to the massive and heterogeneous IoT environment. The IoT will leverage the Internet as a backbone infrastructure for communication and the amount of exchanged data will be overwhelming. Thus, security attacks, such as distributed denial of service (DDoS) attacks, will be severe for the IoT. Furthermore, the IoT will be more vulnerable to security attacks because it will incorporate devices with limited resources and hence with limited security capabilities. Therefore, it would be easier for an attacker to capture such devices and use them in a malicious manner.
The application layer consists of a set of intelligent solution that meets user needs. Users may share information at this level and hence access control, authentication, and key agreement across the heterogeneous devices are mandatory to ensure security and privacy protection [Suo et al., 2012]. Different applications tend to have different security needs and the security solution should adapt to the security requirements of each application domain.

In order to maintain the confidentiality of exchanged information, the communicating devices might use either by-hop or end-to-end encryption [Gang et al., 2011]. In by-hop encryption, each node along the communication path between the communicating nodes decrypts and then encrypts the data before forwarding it to the next node. Thus, a transmission node might perform data aggregation by combining data from several sensor nodes using algebraic or statistical computations before forwarding the aggregated data to the destination node. Although the by-hop encryption provides low latency, high efficiency, and scalability, it allows every transmission node to decrypt and interpret the exchanged data, which requires the use of trustworthy nodes to ensure confidentiality. In end-to-end encryption, transmission nodes forward encrypted data and decryption occur only at the destination node. Thus, it provides high-security protection for exchanged data. However, it cannot protect the destination address and the transmission nodes can no longer perform data aggregation.

Authenticating things before joining the network is necessary to prevent illegal access. Each node should be capable of confirming the identity of other nodes with which it communicates. In order to authenticate things, each thing should be uniquely identifiable. This might be accomplished either using RFID-based identification (or similar identification technologies) or self-description-based identification [Miorandi et al., 2012]. RFID-based identification involves tagging things with RFID tags and the use of appropriate devices to read the tag identifier and retrieve thing description and features from a database. In self-description-based identification, things can communicate directly their identity and available features but require equipping everyday object with storage and communication capabilities. The RFID-based identification is a cheaper solution but requires the reader to access a database to retrieve information about the thing. An identity management system can offer a secure
way to authenticate things. A federated identity management system allows a set of different IoT systems to establish trust relationships with respect to the identity information where the authentication of a thing occurs only once before accessing any of these systems.

Compared to traditional computer networks, IoT systems are more vulnerable to security attacks and they need to survive those attacks given their limited resources. Developing a lightweight and yet effective security solution for the IoT is an open research issue. Moreover, improving software implementation quality is necessary, since it is infeasible to provide a software patch for billions of devices [Roman et al., 2011].

2.3.6 Privacy

In IoT, things will collect information about the environment and the individuals to offer useful services. However, connecting everyday objects to the Internet creates many opportunities for privacy violation. Unlike the traditional Internet where the user controls, to some extent, what, when and where to share private information, IoT devices may collect information about individuals without them even knowing about it. Moreover, privacy issues may arise for people not using the IoT. For example, a surveillance system will take pictures for people visiting the site without their permission. Therefore, in many situations, it will be impossible for people to control the disclosure of their information. Addressing the privacy concerns is crucial to the success and widespread of IoT applications. Therefore, the IoT paradigm must specify the privacy policies for different systems, which should be enforced either by the IoT infrastructure of the individual applications [Stankovic, 2014]. Additionally, it should support privacy negotiation between interacting systems with different privacy policies.

The IoT will collect a tremendous amount of data that will be easily available through remote access. Consequently, an adversary may exploit this to gain access to sensitive information about individuals and groups remotely and anonymously. Attacks on privacy include eavesdropping, traffic analysis, and data mining [Abomhara and Koien, 2014]. Eavesdrop-
ping is the most common and easiest attack that can expose sensitive information to the attacker given that most of the IoT devices will communicate wirelessly. In order to protect data confidentiality, encryption techniques can serve as a countermeasure for this attack. An adversary can use traffic analysis to identify patterns of activities in an IoT system. Data mining technologies enable the discovery of useful information by exploring a large amount of data. However, in wrong hands, it becomes a threat where an attacker can use these tools to infer unanticipated information by eavesdropping and exploring the IoT traffic [Clifton and Marks, 1996].

The IoT must provide users with the ability to control the disclosure of their sensitive information. It should also support transparency where the users can keep track of the usage of their information and be fully aware of entities that accessed their information together with the ability to grant or deny access permissions. Moreover, IoT systems should collect data about individuals anonymously to minimize the risk of violating their privacy. However, some IoT systems may require associating collected data with individuals to serve the purpose of the application. For example, an IoT application might need information about the preferences of a given individual to provide him with personalized recommendations.

The significant decrease in storage cost allows for retained collected information indefinitely. This raises concerns about the right to be forgotten. The IoT systems should support digital forgetting where the system periodically checks for and deletes any collected data or generated information that are no longer of use with respect to the purpose for which it was collected or generated [Atzori et al., 2010]. The system might allow users to define an expiration date for the information they provide.

### 2.3.7 Social Sensing

Given the widespread and popularity of social networks, they have become a massive source of information. Micro-blogging services, like Twitter, allow users to create instant posts covering almost all daily life aspects. Moreover, posts are commonly associated with spatial and
temporal information, which provide valuable contextual information. This has motivated several researchers to leverage social networks in detecting and analyzing events.

In [Sakaki et al., 2010], the authors developed a real-time earthquake detection system that monitors Twitter posts (tweets) and delivers notification promptly for early warning purposes. In order to determine whether a tweet is relevant, the system employs a support vector machine (SVM) [Joachims, 1998] tweet classifier, which uses features such as keywords and number of words in a tweet. Given the location information associated with tweets, the system uses Kalman filters and particle filters to estimate the location of the event [Fox et al., 2003].

In [Takahashi et al., 2011], the authors developed a system that generates a hay fever map by collecting and analyzing tweets. They compared the output of the system with pollen data collected by ordinary physical sensors and found that there was a positive correlation between pollen data and tweet data. Consequently, they have concluded that Twitter can be used as a sensor to detect natural phenomena, at least in some particular areas.

In [Li et al., 2012], the authors developed an event detection and analysis system that crawls Twitter for tweets of potential relation to a particular type of events (e.g. crimes and disaster events), uses a classifier to determine whether a tweet is relevant, and maintains a database of collected tweets together with their temporal and location information. The user can query the system about a particular event by providing a keyword, a spatial range, and a temporal period. In response to a query, the system retrieves relevant tweets from the database, uses a ranking model to rank tweets according to their importance, and visualize the results to the user.

In social sensing, social networks users are virtual sensors and their posts are virtual sensory readings [Sakaki et al., 2010]. The IoT can benefit from the worldwide and the vastly numerous users of social networks by incorporating them into the IoT as social sensors. However, social sensors differ from the ordinary physical sensors in a number of ways. First, social sensors are humans that have different personalities and interests. Therefore, two social sensors
may encounter the same event but act differently. For example, one social sensor may report
the event (i.e. make a post) while the other may simply neglect it. Second, the operation
pattern of a social sensor is unpredictable, where a social sensor may be inoperable due to
being busy or sleeping. Third, the sensory readings (i.e. posts) do not follow a standard
format and a considerable amount of sensory readings is noise. Forth, location information
may be missing because social posts do not always have attached GPS data. Fifth, even
when the location information is present, this does not necessarily imply the location of the
event because the social sensor may be far from the reported event.

Incorporating social sensors in the IoT requires addressing a number of challenges. The
popularity of social networks differ from one geographical region to another and the average
number of sensory readings posted by one social sensor in a particular region may vary
from the others. Thus, applications that use the number of sensory readings as a source of
information must consider the varying density of social sensors from one region to another.
One solution is to explore the distribution of sensory readings over different regions, calculate
a weight for each region, and use this weight to normalize the number of future sensory
readings [Takahashi et al., 2011].

The massive number of instant social posts can overwhelm any system and prevent real-
time responses. Although the use of keywords will retrieve only the potentially relevant
posts, the number of retrieved posts is still too large to analyze. Moreover, many posts
are noise even if they contain the keywords, which can confuse the system and affect its
accuracy [Yerva et al., 2011]. For example, a fire detection system that searches for social
posts containing the keyword fire may retrieves posts about the Kindle Fire tablet or posts
about fire accidents that happened in the past. Text classification techniques can serve as
a preprocessing step to filter out irrelevant sensory readings. Thus, relieve the system from
the unnecessary overload and enhance its accuracy.

Several text classification techniques can be found in the literature [Aggarwal and Zhai, 2012].
Given the high dimensionality of text features and the existence of noisy features, represen-
ing a document by a reduced set of features rather than a stream of letters can help improve both the performance and the accuracy of the classification process. A commonly used approach is to represent a document as a bag-of-words that is a set of words together with their associated frequency in the document. Several variations of the bag-of-words approach have been proposed. For instance, instead of simply using term frequency to indicate the importance of a word, the term frequency-inverse document frequency (tf-idf) numerical statistic is commonly used. The tf-idf can reflect the importance of a word to a document in a collection of documents by diminishing the weight of terms that occur frequently in all documents (e.g. articles) and increasing the weight of terms that occur rarely across documents (i.e. keywords).

Although the bag-of-words based text classifiers have proven to perform well in document classification, they may suffer limitations when applied to social posts for several reasons. First, social posts tend to be short and terms will rarely repeat within the same post. Second, the bag-of-words representation does not consider the order of words and hence loses context information, which becomes significantly important given the short length of social posts. Third, misspellings, shorthands, and slang words are common in social posts. Several researchers proposed different techniques to address those limitations.

In Kontopoulos et al., 2013, the authors proposed the use of ontology-based techniques for sentiment analysis of Twitter posts. Their methodology starts by creating a domain ontology using Formal Concept Analysis (FCA) or ontology learning and then performing sentiment analysis based on concepts and properties included in the ontology.

In Tsur et al., 2013, the authors developed a framework for clustering partially tagged Twitter posts. Their approach starts by creating non-sparse virtual documents, where each document is the result of concatenating all tweets that contain one of a given set of tags. Each document is represented as a feature vector using the bag-of-words approach then the documents are clustered using a k-mean algorithm. A new tweet containing a given tag is assigned to the same cluster to which the virtual document for that tag was assigned.
In [Dilrukshi et al., 2013], the authors used support vector machine (SVM) for news classification. They collected and manually classified tweets into 12 groups covering main areas of general news providers. They used the bag-of-words approach to extract features from tweets and removed low (noise) and high (common) frequency words. Results have show a variation in the classification performance between different groups. Although the classification performance was relatively good, this tend to be a result of the selected case study. The tweets contained news headlines, which are commonly formal and representative.

Given that company names are often ambiguous, the authors in [Yerva et al., 2012] presented a tweet classification technique to determine whether a tweet is related to a particular company. They started by creating a profile for each company (i.e. a set of weighted words) using a variety of source (e.g. companys homepage). They used the bag-of-words approach to represent tweets and the Nave Bayes classifier to compare tweets with the profiles. In order to maintain an active profile, words co-occurring with profile keywords in a tweet are added to that profile.

In [Sriram et al., 2010], the authors proposed a technique to classify tweets into general categories (news, events, opinions, deals, and private messages) using discriminative features extracted from authors profile and the text of the tweet. The technique makes use of a small set of assumption, where certain features can reflect to which class the tweet belongs. For instance, existence of a currency symbol within the tweet text may indicate a deal. Furthermore, the technique assumes that majority of tweets from the same author tend to belong to a limited number of categories. Although the proposed approach performed well in classifying tweets to the mentioned general categories, noise removal techniques are necessary because noisy data can degrade the performance.

In [Nishida et al., 2011], the authors proposed a different approach that uses data compression rather than learning techniques for tweet classification. Given two sets of positive and negative samples for a given topic, a tweet is classified as related to that topic if it gives better compressibility with positive samples than with negative samples. In order to evaluate
the compressibility of a tweet with a set of samples, a compression algorithm is applied twice to compress the samples before and after appending the tweet. The compressibility of the tweet with that set of samples is the difference in size between the two compression results. Unlike the bag-of-words approach, this method does not require dividing a tweet into words, which makes suitable for languages that do not use whitespaces to delimit words, such as Japanese. Moreover, it avoids the problems associated with the bag-of-words approach such as losing word context and the need to handle misspelled and slang words.

An ordinary physical sensor often has a limited sensing range and hence the location of the sensor implies the location of the sensed phenomena. In case of social sensors, it is possible to infer the event location from the text of the post if it contains explicit location name. Under the assumption that the social sensor is near to the reported event, the GPS data attached to the sensor reading can serve as the location of the reported event. If the GPS data is missing, location information can be obtained from the user profile. However, user profiles are not always up to date and may have invalid or no location information. As a final resort, it is possible to estimate the location of a social sensor by exploring his historical posts and/or social relationships in the social network [Li et al., 2012]. Historical posts may have GPS tags or mention explicit locations. However, estimation based on historical posts tends to be inaccurate, especially for social sensors that move frequently. Estimating the location of a social sensor based on the locations of friends assumes that those friends live in nearby, which is not always the case.

Jurgens [Jurgens, 2013] proposed a method for location estimation of Twitter users based on the assumption that individuals often form social relationships with those located nearby. Their method extends the label propagation algorithm, which is a semi-supervised iterative algorithm that can infer labels for nodes in a network given a small set of labeled nodes. Instead of labeling nodes with the most frequent label in their neighbors as in the original algorithm, their spatial label propagation algorithm used other methods for selecting a location from the list of neighbors locations, such as the geometric median.
2.3.8 Social Things

Social networks provide a paradigm where people can collaborate and share information. Social things refer to the incorporation of things into the social networks used by humans or the formation of social networks dedicated only to things. This can enable spatially distributed things to form a community, where they collaborate and share information much similar to what people do. Social things should have the ability to share its local information with other community members, integrate local information with information obtained from the community to solve particular problems, and interpret the exchanged information regardless of used formats [Vazquez and Lopez-De-Ipina, 2008].

In [Baquer and Kamal, 2009], the authors proposed the use of micro-blogging to publish and share locally measured sensory readings. They designed a model, namely S-Sensors, to connect the environment with end-users via integrating wireless sensor networks with Twitter. Their design has four components: sensor nodes, base station, tweet messages, and Twitter portal. Sensor nodes sense the user environment to capture readings related to the phenomena of interest. The base stations, which runs a Twitter client, collects sensory readings, possibly perform data aggregation to reduce the amount of communicated sensory information, and sends tweet messages depicting the status of the environment.

In [Guinard et al., 2010], the authors developed a system that relays on social networks to allow people to share and use web-enabled things. The system leverages social networks to authenticate users via delegated authentication and to retrieve lists of social relationships to serve as trusted connections. The sharing of a thing occurs in three phases. First, the owner logs into the system using one of his social networks credentials. Second, the system crawls the smart thing to identify sharable functionalities. Finally, the owner generates the access control list that defines which of the trusted connections can use what functionality. The system can support composite services and mashups by providing the ability to share and use smart things. However, scalability and security issues in that system still need further investigation.
In [Kranz et al., 2010], the authors demonstrated the potential of combining technological networks, such as WSNs, and social networks to accomplish dedicated tasks collaboratively. They introduced Cognitive Office as a smart environment, where the sensors can send posts to Twitter about detected events and the devices interested in those events can listen to the tweets in a publisher/subscriber manner. The system monitors objects, such as shelf and coffee mug, to observe usage patterns and provide services to the users.

Social networks have proven to be efficient in handling the relationships among humans and can provide many benefits to the IoT. The difference in terms of characteristics and goals between things and humans may propose the need for social networks dedicated to things. Such networks can provide appropriate profile structures that allow for proper digital representation of things and their capabilities, novel types of social relationships between things that cover all possible types of interactions, and techniques to discover and interact with other things [Atzori et al., 2014b]. The effort towards the creation of social things is still in its infancy. Most of the work focused on using social networks of humans either to share things under the control of their owners or to incorporate things that can report their sensory readings as social posts. In the social-oriented vision of the IoT, things will be able to establish social relationships autonomously and independent of the social relationships of their owners. They will form collaboration communities, where they share information and services. Those communities allow for the creation of composite services and complex applications in a scalable, trustful, and efficient manner [Atzori et al., 2014a], where the collaboration occurs only between things having trustful connections (i.e. social relationships). Thus, things need to navigate only through their trustful connections to discover other things that they can collaborate with rather than using Internet discovery tools, which cannot scale to cover billions of IoT devices.

Social Internet of Things (SIoT) [Ortiz et al., 2014] paradigm refers to an ecosystem where people and smart things can interact within a social framework that allows for offering a variety of services and applications using web technologies. The IoT mainly supports two models of interaction: human-to-human interaction inherited from the Internet and thing-
to-thing interaction where the humans act only as consumers. The SIoT paradigm focuses on the human-to-thing interaction where humans act as both consumers and producers of information. In SIoT, humans can establish relationships with social things and can participate in creating services. Thus, humans and things become cooperating nodes in the network leveraging the three models of interaction: human-to-human, thing-to-thing, and human-to-thing.

2.3.9 Multi-Sensor Data Fusion

Multi-sensor data fusion is the process of combining sensor data or data-driven from sensory data into a common representational format [Mitchell, 2012]. On the other hand, multi-sensor integration refers to the synergistic use of sensory information obtained from multiple sensors to assist in the accomplishment of a given task without actually fusing the sensory information into one representational format [Luo and Kay, 1989]. For example, using information obtained from one sensor in a given step to guide the operation of other sensors in subsequent steps.

Fusing or integrating redundant data provided by multiple sensors can improve the accuracy of inferences as well as the reliability in case of sensor error or failure. Moreover, the complementary information obtained from multiple sensors allows for perceiving features and making inferences that are not achievable using data collected by each sensor separately [Hall and Llinas, 1997].

A data fusion model describes the set of steps that a data fusion system should undertake. One of the most widely used data fusion models is the JDL model [White et al., 1988] proposed by the Joint Directors of Laboratories (JDL) data fusion sub-panel, which was established by the US Department of Defense. Originally, its main purpose was to aid in the development of military applications. In [Steinberg et al., 1999] the authors revised the JDL model to include five levels: sub-object data assessment, object assessment, situation assessment, impact assessment, and process refinement.
The Thompoulos model [Thomopoulos, 1990] uses a three level data processing architecture for data fusion. The three levels are the signal level, the evidential processing level, and the dynamic level. At the signal fusion level, the fusion occurs through heuristic rules, correlation, or learning due to the lack of a mathematical model that describes the phenomenon of interest. At the evidence fusion level, the fusion occur based on a statistical model that describes the phenomena. The statistical model may is not necessarily precise and may include fuzziness to allow for incomplete or inconclusive evidence. At the dynamic fusion level, fusion occur with the aid of a mathematical model that describes the phenomena. The interconnections among the three levels can be adapted according to the fusion problem at hand, the fusion objectives, and the characteristics of the used sensors.

The Pau model [Pau, 1988] decomposes the data fusion process into five steps representing basic problems that must be solved in sequence before the final fused output is established. The five steps are alignment process, association process, attribute data fusion, analysis data fusion, and fusing representations.

The Waterfall model [Bedworth, 1994] consists of six modules: sensing, signal processing, feature extraction, pattern processing, situation assessment, and decision-making. The decision-making module continuously updates the multi-sensor system with feedback information to adapt the system (e.g. re-calibrate the sensors).

In [Dasarathy, 1997], the authors proposed five categories of fusion with respect to a three-level fusion hierarchy: data, feature, and decision fusion. The first category involves fusing input data to produce fused output data. The second category involves combining data from multiple sensors to derive some feature of the phenomenon of interest. The third category is feature fusion where both inputs and outputs of the fusion process are features. The fourth category is the fusion of features produce a decision output. The fifth category is decision fusion where both the inputs and the outputs are decisions.

The Omnibus model [Bedworth and O’Brien, 2000] follows the general structure of the Boyd control loop [Boyd, 1987] but uses the finer definition of the Waterfall model. The Boyd (or
OODA) loop consists of a closed loop of four phases: observe, orientate, decide, and act. The Omnibus model consists of four basic modules corresponding to the four phases of the OODA loop. The “sensing and signal processing module” corresponds to the “observe phase”. The “feature extraction and pattern processing module” corresponds to the “orientate phase”. The “context processing and decision-making module” corresponds to the “decide phase”. Finally, the “resource tasking and control module” corresponds to the “act phase”.

Incorporating social sensors into a data fusion system introduces several challenges emerging from their nature and characteristics. Unlike sensory data and information stored in structured databases, the data obtained from social sensors tend to be noisy and does not follow any standards or specific formats. Consequently, in order to fuse social and sensory data, there is a need for a data fusion model that considers the peculiarities of social sensors as an unconventional data source. The model needs to consider tasks such as noise filtering, text mining, data association, semantic alignment, and location estimation of the data source.

2.3.10 Thing Discovery

Several discovery service protocols can be found in the literature. One popular protocol is the Universal Plug and Play (UPnP) protocol, which relies on peer-to-peer communications between devices and control points [Miller et al., 2001]. The architecture supports addressing, discovery, device/service description, control, event notification, and presentation. It relies on IP addressing, where a device is assigned an address through the Dynamic Host Configuration Protocol (DHCP) [Droms, 1997]. If a DHCP server is not available, the device uses an automatic IP addressing technique as described in the UPnP specifications. The UPnP discovery protocol is based on the Simple Service Discovery Protocol (SSDP) [Cai et al., 1999]. It allows the devices to advertise their services to the control points through discovery messages. Devices can later send messages to cancel advertisements or renew them before expiration. Control points can passively listen to the discovery messages or actively search for devices of interest on the network. The discovery message
contains a URL for the device description. Device description contains URLs for service descriptions. Service description includes a list of supported actions and a list of state variables. Both device and service descriptions use standard UPnP templates expressed in eXtensible Markup Language (XML). Control points use GET requests to query those URLs and receive Hypertext Transfer Protocol (HTTP) response.

For control points to interact with devices, they need mechanisms to send commands to those devices and to receive notifications about changes in their state. The service description contains a control URL to which a control point can send a message to invoke an action. Control messages are expressed in XML using the Simple Object Access Protocol (SOAP). Control devices send control messages via HTTP requests and receive results or errors via HTTP responses. Similarly, the service description contains a subscription URL that allows subscribers to receive updates upon change in service state. A control point can send a subscription message to the subscription URL and provide a delivery URL to which event notifications should be sent. Subscribing to a subset of the state variables is not supported and subscriptions can be canceled or renewed before expiration by sending the appropriate messages to the subscription URL. Notifications are sent via HTTP in the form of an XML structure specifying the names of one or more state variables and their new values. Besides the functional interface provided by control and event notification mechanisms, a device may provide a web-based interface in the form of a HyperText Markup Language (HTML) page. A control point can retrieve that page by sending an HTTP GET request to the page URL provided in the device description.

The UPnP architecture is designed for TCP/IP networks only and relies on standard Internet protocols to support peer-to-peer discovery and interaction between devices and control points. However, supporting discovery service through message multicasting can result in exponentially increasing traffic as the number of devices and control points increase, which can overwhelm the network bandwidth. Moreover, the use of XML and HTTP requests for communication between devices and control points is not suitable for constrained IoT devices.
The Service Location Protocol (SLP) is another service discovery protocol that relies on three agents to support resource discovery: user agents, service agents, and directory agents [Guttman, 1999]. User agents perform service discovery on behalf of client software, service agents advertise the location and attributes on behalf of services, and directory agents are repositories for service information. User agents and service agents use multicast messages to discover directory agents on the network. In active discovery, user agents and service agents multicast SLP requests to the network while in passive discovery, directory agents multicast advertisements for their services periodically. Once a directory agent is discovered, user agents can unicast their requests to it. Service agents can as well advertise their service information to the directory agent in the form of a service location (URL) and an optional list of key-value pairs representing service attributes. In the absence of a directory service, a user agent repeatedly multicast its request to the network. Service agents listen to these multicast requests and unicast responses to the user agent if they have the requested service.

Multicast discovery is required only to discover directory agents or to discover services in the absence of directory agents. The presence of directory agents can improve service discovery by reducing the involved network traffic and allowing user agents to receive faster responses as they need to unicast their requests to the directory agent compared to multicasting them to the network. Moreover, multicast discovery for directory agents can be avoided if their addresses can be distributed through the DHCP servers when requested by user agents and service agents using DHCP Options for SLP [Perkins and Guttman, 1999]. By default, SLP agents send all messages in User Datagram Protocol (UDP) datagrams and use Transport Control Protocol (TCP) only when a message cannot fit in a single datagram. SLP uses multicast convergence algorithm to avoid sending multicast messages to agents that has already responded.

SLP relies on either multicast discovery or DHCP for its own configuration. Therefore, it does not scale to the Internet. Although SLP does not provide a mechanism for access control, it supports service authentication, where services can be digitally signed using public-key cryptography. However, this requires the public keys of the services to be installed on
every user agent, which requires prior configuration that violates the original purpose of SLP. Moreover, the authentication mechanism has leaks that makes the protocol vulnerable to attacks. For example, an attacker can eavesdropping on messages between SLP agents before resending a recorded message at a later time. Thus, the attacker might be able to spoof a legitimate agent. In [Vettorello et al., 2001], the authors proposed a modification to the SLP specification, where the agents cache the timestamp of the received messages and accept only messages with a larger timestamp than that of the previous message. This technique might help authenticating the exchanged messages and the URLs they contain. However, IP addresses can be spoofed and hence the authenticity of a URL does not guarantee the authenticity of the responding device.

The Constrained Application Protocol (CoAP) [Shelby et al., 2014] is designed for lightweight machine-to-machine (M2M) communication between resource-constrained devices. It supports a client/server interaction model, where a CoAP endpoint can act as both a client or a server. CoAP supports the Representational State Transfer (REST) architecture by providing the basic methods of GET, POST, PUT, and DELETE, which are easily mapped to/from HTTP for integration with the web. Two main mechanisms have been proposed for service and resource discovery in CoAP: direct resource discovery and directory-based resource discovery.

In direct CoAP resource discovery [Shelby, 2012], the client issues a GET request to the well-known URI of the server (/well-known/core) and the server replies with the URI of all of its discoverable resources. If the server address is known, the client can send a unicast request to the target server. Otherwise, the client can send a multicast request if IP multicast is supported within the network and receive multiple responses from different servers on that network. A client can query for specific types of CoAP resources by specifying one or more search parameters in the query string listed as parameter=value pairs. A server can decide which of its available resources are discoverable. This mechanism allows for direct communication between CoAP clients and servers. However, this requires a client to know the URI of the server to be queried. Relying on static configuration of server addresses
is not desirable in a dynamic environment, where devices can join and leave the network frequently. On the other hand, the use of multicasting discovery is not reliable because there is no guarantee that a request will reach all of the intended servers and therefore the client may not obtain complete information. Performing serial unicast to server addresses obtained from the network layer can be inefficient, especially for constrained devices.

In directory-based CoAP resource discovery [Shelby et al., 2013], a resource directory (RD) stores the resource descriptions provided by the CoAP servers and allows lookups for these resources by CoAP clients. The RD supports interfaces for devices to register, maintain, lookup, and remove resource descriptions. Compared to direct discovery, clients can discover any required resource with a single query to the RD. However, for a device to perform registration or lookup, it needs to know how to reach the RD. This can be achieved in many ways such as using a default location, assigning an anycast address to the RD, using DHCP, or by discovering the RD using the CoRE Link Format [Shelby, 2012]. A server can register its resource within the RD via a POST request. If the registration is successful, the RD responds with the path of the registered resource. The server can use this path later to either update or delete the resource. The RD may also proactively discover the resources of different servers if the servers make them available through the well-known URI. In order to maintain the RD entries, the RD deletes resources when their registration lifetime expires. Moreover, the RD can proactively validate its entries by querying the registered servers before either deleting or updating their resource descriptions. Once the resources are registered within the RD, clients can start querying to obtain information about resources of interest.

In Domain Name System Service Discovery (DNS-SD) [Cheshire and Krochmal, 2013a], the DNS resource records are named and structured to facilitate service discovery without altering the DNS protocol specifications. A DNS record is a mapping between two parameters. The DNS-SD utilizes the pointer (PTR), service locator (SRV), IPv6 address (AAAA), and text (TXT) records. The PTR record maps a service type in a given domain to a service instance name of that type in that domain. The SRV record maps a service instance name to the service URI. The AAAA record maps a service instance name to the IPv6 address. The
TXT record maps a service instance name to a detailed description of the service instance containing information about its resources and attributes. Using standard DNS queries, a client can discover a list of names instances of the desired service type within the desired domain. The client queries for the PTR record specifying the service type and domain to obtain the service instance name before querying for the SRV record to obtain the host and port information of the service endpoint. Finally, the client queries for the TXT record to obtain service description. DNS-based service discovery can be performed in a distributed manner using multicasting or in a centralized manner by unicasting to a central server.

In distributed DNS-SD, no central DNS server is available and queries are multicasted to discover the services as in multicast DNS (mDNS) [Cheshire and Krochmal, 2013b] and extended multicast DNS (xmDNS) [Lynn and Sturek, 2012]. The mDNS provides DNS-like operations on the local network in the absence of a conventional DNS server. Consequently, it requires little or no administration or configuration. Moreover, it is robust against infrastructure failures and suitable for infrastructureless networks. Instead of querying a DNS server through unicast, a client can query the local devices directly through multicast. Using standard DNS queries, devices can publish information about the services they provide using mDNS advertisements. An advertisement can be a PTR record specifying service type and name, a SRV record specifying host name and port, an AAAA record specifying an IPv6 address, or a TXT record specifying service description.

In centralized DNS-SD, a DNS server is available and it stores service descriptions for the devices. A device registers its services within the DNS server through a DNS registration message sent directly to the unicast address of the server. If DNS server address is unknown, the device may discover the address through browsing the well-known service type or use multicasting as in mDNS to publish the registration message. If the DNS-SD server is in a different subnet, global DNS-SD should be used to automatically discover the address of the remote server. A client can lookup the available services through DNS queries sent to the DNS-SD server. Compared to the distributed DNS-SD, the client will receive a single response from the DNS-SD server containing all the services that matches the query.
Table 2.1: Comparison of discovery services.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Transport</th>
<th>Complexity</th>
<th>Topology</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPnP</td>
<td>IP-based</td>
<td>High</td>
<td>P2P</td>
</tr>
<tr>
<td>SLP with DA</td>
<td>IP-based</td>
<td>High</td>
<td>Centralized</td>
</tr>
<tr>
<td>SLP without DA</td>
<td>IP-based</td>
<td>High</td>
<td>P2P</td>
</tr>
<tr>
<td>CoAP with RD</td>
<td>IP-based</td>
<td>Low</td>
<td>Centralized</td>
</tr>
<tr>
<td>CoAP without RD</td>
<td>IP-based</td>
<td>Low</td>
<td>Distributed</td>
</tr>
<tr>
<td>DNS-SD with a server</td>
<td>IP-based</td>
<td>Low</td>
<td>Centralized</td>
</tr>
<tr>
<td>DNS-SD without a server</td>
<td>IP-based</td>
<td>Low</td>
<td>Distributed</td>
</tr>
<tr>
<td>Zigbee</td>
<td>Non-IP</td>
<td>Low</td>
<td>Hybrid</td>
</tr>
</tbody>
</table>

In the context of supply chains, the EPCglobal network allows trading partners to share product data. Products are identified using Electronic Product Code (EPC), which is universally unique for any physical item, for all time. This allows for tracking each object worldwide by means of automatic identification technologies such as Radio-Frequency Identification (RFID) as well as barcodes. The EPCglobal network architecture framework incorporates three main components: the EPC Information Services (EPCIS), the Object Name Services (ONS), and the Discovery Services [Armenio et al., 2007]. The EPCIS are the repositories at each company’s site that support storage and retrieval of information about different items in the supply chain. The ONS uses Domain Name System (DNS) infrastructure to retrieve information about products, where the EPC is converted to a domain name and the query result is a valid DNS resource record [GS1, 2013]. The discovery services provides a method to establishing contact between the querying client and the EPCIS that hold information about the EPC of interest.

The Building Radio frequency IDentification for the Global Environment (BRIDGE) project [BRIDGE, 2009] introduced four models for implementing the discovery service including directory-of-resources, directory-of-clients, notification-of-resources, and notification-of-clients.
In the directory-of-resources model, each EPCIS publishes information about the availability of EPC-related data into the discovery service. The client queries the discovery service with an EPC to receive the identity of relevant EPCIS before querying those EPCIS for detailed information about the EPC of interest. In the directory-of-clients model, the client publishes its interest in certain EPC numbers to the discovery service. Each EPCIS looks up the keys in the discovery service to determine the clients it can serve before informing them of its identity so that the clients can query it for detailed information about EPC numbers of interest. In the notification-of-resources model, the clients subscribe to the discovery service for notification about the EPC number of interest. When an EPCIS publishes its availability to the discovery service along with the EPC numbers it holds data about, the discovery service notifies the clients interested in those EPC numbers so that the clients can query that EPCIS for detailed information about EPC numbers of interest. In the notification-of-clients model, each EPCIS publishes its availability together with the EPC number it holds information about. When a client publishes its interest in certain EPC numbers, the discovery service informs the relevant EPCIS before they contact the client informing it about their identity and the EPC numbers they hold information about so that the client can query them for detailed information about the EPC number of interest.

In [Kürschner et al., 2008], the authors described a design of discovery service for the EPC-global network based on a set of requirements that they derived from interviews and literature research. Compared to the directory-lookup design, where the data and access rights are stored at the discovery service level, their query-relay design forwards a company’s query about a given EPC to the relevant EPCIS, where the access rights are checked locally at that EPCIS before answering the initial query directly to the querying company. This way companies are in control of their data, which encourages them to participant in the network.

In [Paganelli and Parlanti, 2012], the authors presented a distributed Discovery Service using a peer-to-peer approach. Similar to [Manzanares-Lopez et al., 2011], their framework relies on a distributed hash table (DHT) technique. In addition to simple ID-based queries, their framework supports multi-attribute and range queries. In [Liu et al., 2013], a dis-
distributed resource discovery architecture (DRD) for M2M applications is proposed. They proposed to hash the MAC address of the device to generate unique endpoint name. However, a MAC address can be spoofed by software which can lead to duplicates. The DRD consists mainly of a resource reregistration component for storing resource description and a resource discovery component that caches the most recently queried resource values to reduce discovery time. Using a peer-to-peer overlay distributes the workload and avoids single point of failure. The authors of [Cirani et al., 2014] proposed a self-configuring peer-to-peer service discovery architecture. Their architecture relies on IoT gateway to keep track of things joining or leaving its network and to support proxy functionality as well as resource and service discovery. The IoT gateways are interconnected through two peer-to-peer overlays: the distributed location service (DLS) and the distributed geographic table (DGT). The combination of DLS and DGT allows for large-scale service discovery that is scalable, robust, and self-configurable while enabling service and resource discovery on a geographical basis.

In [Datta and Bonnet, 2015], the authors presented a list of consumer centric requirements based on which they presented a search engine based resource discovery framework. Their framework is composed of three layers: the proxy layer, the discovery layer, and the service enablement layer. The proxy layer allows for communicating with devices regardless of the communication technologies they use. The discovery layer provides a configuration storage for devices, indexing of devices, and a search engine. The service enablement layer exposes the functionalities of the discovery layer through RESTful web services and includes an access control service that restricts the discovery to the resources that the consumer has access to.

### 2.3.11 Ontologies

Several semantic approaches have been proposed in the literature to address the challenges related to interoperability. Researchers have proposed several ontologies to describe different aspects in the IoT domain. Ontology is defined as “an explicit formal specification of
a shared conceptualization” [Ming and Jie, 2002]. It provides a well-founded mechanism to represent and exchange semantic meaning using a formal description of a set of concepts and the relations between them [Ye et al., 2007]. Existing ontologies try to tackle various problems related to heterogeneity such as thing discovery [Bermudez-Edo et al., 2016], data description [Gyrard et al., 2014], thing capabilities [Xue et al., 2015], data access and sharing [Shi et al., 2012], and extensibility [Balaji et al., 2016]. In [Ye et al., 2007], the authors proposed two classifications of ontologies based on expressiveness as light-weight or heavy-weight and based on generality as generic, domain-specific, or application-specific.

Several approaches have been proposed to evaluate an ontology. Validation tools such as OntoCheck [Schober et al., 2012] can help ontology creators to verify ontology naming conventions and metadata completeness. Metric-based approaches can provide a quantitative measure of ontology quality [Tartir et al., 2010]. Schema metrics address the design of the ontology by providing indicators of the richness, width, depth, and inheritance of an ontology schema. Instance metrics evaluate the placement of instance data within the ontology and measures the utilization of each schema class providing indicators of the effectiveness of the ontology design. In [Brank et al., 2005], the authors classified ontology evaluation approaches into four categories: (1) approaches based on comparing the ontology with another, (2) approaches based on using the ontology in an application and evaluating the results, (3) approaches based on comparing the ontology with a source of data about the domain to be covered, and (4) approaches based on manual assessment by human experts against a set of predefined criteria. The authors in [Staab and Studer, 2010] identified eight parameters to assess the quality of an ontology, which are accuracy, adaptability, clarity, completeness, computational efficiency, conciseness, consistency, and organizational fitness.

The Semantic Sensor Network (SSN) ontology [Compton et al., 2012] was built by the World Wide Web Consortium (W3C) for describing the sensors and the data they collect. SSN is one of the most popular ontologies for addressing heterogeneity problems associated with sensor discovery and data collection. However, it has limited concepts to support spacial and temporal association of collected data with sensors [Bajaj et al., 2017]. The SSN ontology
provides a detailed description with concepts and properties that enables it to support a wide range of applications. However, it is considered a heavy-weight ontology as it includes non-essential components for many use cases.

The IoT-Lite ontology [Bermudez-Edo et al., 2016] is a lightweight instantiation of the SSN ontology to describe IoT resources, entities, and services while remaining suitable for constrained IoT environments. It can be extended and combined with other ontologies to represent IoT concepts in different domains. In [Bermudez-Edo et al., 2017], the authors defined a set of guidelines for developing scalable ontologies that they have followed in their design of IoT-Lite. They evaluated IoT-lite on scalability and complexity against another instantiation of SSN, IoT-A [Bauer et al., 2013], which provides a detailed model to define concepts for devices, services, and context. Evaluation results demonstrate that IoT-Lite performs better than IoT-A, in terms of memory requirements, computational time and Round Trip Time (RTT) for a query-response. IoT-Lite use three classes to describe IoT concepts, which are object, system, and service. It classifies IoT devices into sensing, actuating, and tag devices. Figure 2.1 shows the concepts of the ontology and the main relationships between them.

Location-awareness is a crucial subject in the IoT, where the location information can be used to provide adequate services or to adapt to changes in the physical environment [Schuhmann et al., 2008]. The Global Positioning System (GPS) can provide location information in terms of longitude and latitude. However, common GPS receivers do not work indoor due to poor signal strength caused by construction materials, multipath effects, and limited on-device computation power [Nirjon et al., 2014]. The IoT-Lite ontology uses geo:Point to specify the location of a device in terms of longitude, latitude, and altitude. In addition, it defines two properties, iot-lite:relativeLocation to specify a location not provided by geo-coordinates (e.g. building A, city of New York) and iot-lite:altRelative to specify altitude not provide by the coordinate altitude (e.g. floor 2).

1https://www.w3.org/2003/01/geo/
Several ontologies designed for SBEs use location as a unique concept to define an indoor space (e.g. a building, a room, a passageway) \cite{Wang2004, Huq2007}. However, this introduces a significant weakness in location awareness due to the imprecision, insufficiency and/or ambiguity of location information. In \cite{Abdulrazak2010}, the authors introduced the notion of referential localization. Unlike absolute localization, the referential localization considers the location of objects in relation to other objects taking the form of (upTo, downTo, behindTo, inFrontTo, nearTo, inTo, Xpositive, Xnegative), where Xpositive and Xnegative represent the right and left side of the object according the angle with which the object is perceived. Their goal is to provide additional information about the physical arrangement of objects in space that can enrich location-aware applications.
Chapter 3

Smart Built Environments (SBEs)

The vision driving SBEs is described in many different ways using a variety of terms. Some of those terms can be considered synonyms to the term “smart built environments” while others describe different aspects and technologies inspiring the vision of SBEs. Among those terms are ubiquitous computing, pervasive computing, ambient intelligence, intelligent environment, intelligent spaces, smart environments, and the Internet of Things (IoT).

Ubiquitous computing refers to the creation of a computing environment that enables people to get information and to connect to networks anywhere at anytime [Weiser, 1991]. The term “pervasive computing” refers as well to environments that are saturated with computing and communication capability [Satyanarayanan, 2001]. Inspired by those definitions, an SBE should incorporate computing and communication capabilities into the physical space. This should enable the users to connect to and get information about different physical objects in that space.

Ambient Intelligence (AmI) refers to sensitive and adaptive electronic environments that respond to the actions of persons and objects and cater to their needs [Aarts and Wichert, 2009]. According to this definition, an SBE should have sensing capabilities in order to keep track of persons and objects in presence as well as actuation capabilities in order to fulfill their
needs that may require changing the physical state of the environment. Moreover, being adaptive requires an SBE to be aware of the context. Context awareness is discussed in section 3.3. The IoT is one of the terms related to SBEs. In fact, developing SBEs is one of the several applications of the IoT.

Several definitions related to SBEs can be found in the literature. Researchers use several terms, such as intelligent environments, smart environments, intelligent spaces, and iSpaces, to refer to SBEs. Cook and Das [Cook and Das, 2004] described a smart environment as “a small world where different kinds of smart devices are continuously working to make inhabitants’ lives more comfortable”. They defined “smartness” or “intelligence” as “the ability to autonomously acquire and apply knowledge” and “environment” as a term referring to “our surroundings”. Based on that, they defined a smart environment as one that is able to acquire and apply knowledge about the environment and also to adapt to its inhabitants in order to improve their experience in the environment. In [Augusto et al., 2013], the authors defined an intelligent environment as a physical space with numerous networked controllers controlling different aspects of that space. According to their definition, those controllers are orchestrated by self-programming preemptive processes in order to create an interactive holistic functionality that enhances occupants experiences. In [Steventon and Wright, 2006], intelligent spaces, or iSpaces, are described as spaces with links between the physical objects and the digital world where the information about the physical world can be used to augment human functionality and experience. In [Lee and Hashimoto, 2002], the authors defined an intelligent space as “rooms or areas that are equipped with sensors, which enable the spaces to perceive and understand what is happening in them”. In [Buckman et al., 2014], the authors have identified three main drivers of building progression (efficiency, longevity, and comfort) and four methods (intelligence, control, enterprise, and material and construction). Building automation aims at minimizing user interaction while intelligent buildings focus on gathering information and responding to it autonomously. More recently, the term “smart buildings” is gaining popularity in the literature. It refers to buildings that account for not only intelligence but also enterprise, control, and materials and construction [Buckman et al., 2014].
Inspired by the different definitions proposed for “SBEs” and the related terms, we provide a list of requirements that a physical space needs to meet to be considered an SBE.

- **Computation.** The physical space needs to have computing capability. This can be achieved by incorporating computing devices into space and/or embedding computing elements into the physical objects.

- **Communication.** the computing elements incorporated into a physical space should be uniquely identifiable and accessible through a communication network that allows them to exchange data with each other as well as with the users.

- **Sensing/actuation.** The physical space should have the ability to collect data about its state through embedded sensors as well as the ability to change that state through embedded actuators.

- **Intelligence.** The physical space should have the ability to autonomously acquire and apply knowledge. Knowledge can be inferred from a variety of data sources such as sensors, social networks, streaming services, and other databases. The ability to apply knowledge allows a physical space to adapt to the current context and support user tasks by taking autonomous actions via actuators or providing recommendations to the users.

- **Control.** The physical space should provide its users with means of control. This allows a user to interact explicitly and override autonomous actions if necessary.

Incorporating computation, communication, and sensing/actuation capabilities into a physical space can enhance the user experience by providing new functionalities or enhancing the current ones. However, supporting those capabilities is not enough to claim a physical space as a “smart” one. Those capabilities are just the infrastructure for an SBE but they do not provide any smartness on their own. The term “smart” implies the ability to take intelligent actions. Therefore, for a physical space to be considered smart, it must have the
ability to adapt to the current context. Relying solely on autonomous actions to address user needs may not be feasible or achievable given the current technologies. Moreover, users may receive autonomous actions as patronizing. Therefore, an SBE should support means of control allowing its users to interfere and take actions to address their needs. A number of proposed designs and prototypes for building automation, intelligent buildings, and smart buildings can be found in the literature [Wong et al., 2005, Alaa et al., 2017].

3.1 IoT-Enabled SBEs

For decades, SBE research has been an active topic with several proposed designs that can be found in the literature. Early SBEs were often closed systems operating their own standards and custom protocols. Incorporating IoT technologies into SBEs provides an opportunity to create open systems that can connect to the Internet and benefit from the cloud. An IoT-enabled SBE incorporates a set of things, which are devices with computation and communication capabilities as well as sensing and/or actuation capabilities. Things can sense their surroundings and/or take autonomous actions to change the state of the physical environment.

The IoT is still in its infancy with several challenges yet to be addressed. Interoperability remains an issue due to the lack of widely adopted standards. The current state of the IoT is similar to the state of online services in the 1980s and early 1990s such as CompuServe and AOL, which were closed networks that did not interoperate [Rowland et al., 2015]. Standardization can help turning the IoT into an open network, where different things can interoperate with common standards and protocols.
3.2 User Interaction

Humans-Computer Interaction (HCI) is the study of the way in which computer technology influences human work and activities [Dix, 2009]. According to the Association for Computing Machinery (ACM), HCI is concerned with the design, evaluation, and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them [Hewett et al., 1992]. HCI has emerged in a world of desktop computers, where the interaction usually involved a single user using a single device. However, user interaction in IoT-enabled SBEs is a whole different story, where multiple users may interact simultaneously with multiple things to accomplish varying tasks.

In traditional HCI, humans usually play the active role in the interaction whereas computers take a passive role. Humans feed computers with data and commands while computers just react to those commands. However, in an IoT-enabled SBE, things can play an active role in the interaction as well. They can collect data and take actions autonomously. Consequently, a human may interact with an SBE spontaneously. This leads to two models of user interaction in SBEs, explicit and implicit [Schmidt, 2000].

Explicit user interaction refers to the traditional interaction model where the user intentionally interacts with specific things in the SBE by cognitively providing inputs to those things. In contrast, implicit user interaction refers to the absence of cognitive effort by the user to provide input during the interaction [Kelly and Teevan, 2003]. Table 3.1 shows the two interaction models. Supporting explicit user interaction in an SBE requires providing a user interface through which the user can cognitively provide inputs and/or receive feedback. On the other hand, implicit user interaction does not require a user interface. Instead, the things will autonomously collect data about the current context and use it to take autonomous actions without explicit human intervention.
Table 3.1: User interaction models in IoT-enabled SBEs.

<table>
<thead>
<tr>
<th>Model</th>
<th>Human role</th>
<th>Thing role</th>
<th>User Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit</td>
<td>Active</td>
<td>Passive</td>
<td>Required</td>
</tr>
<tr>
<td></td>
<td>Active</td>
<td>Active</td>
<td>Required</td>
</tr>
<tr>
<td>Implicit</td>
<td>Passive</td>
<td>Active</td>
<td>Not required</td>
</tr>
</tbody>
</table>

### 3.3 Context-Aware Interaction

The term “context-aware” was first introduced in [Schilit and Theimer, 1994], where the authors defined context-aware computing as “the ability of a mobile users application to discover and react to changes in the environment they are situated in”. They used the term “context” to refer to the location of use, the nearby people and objects, and the changes to those objects over time. However, context should not be restricted to these examples. In [Dey, 2001], the author defined context as “any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves”. This operational definition describes the context in general without restricting it to specific examples. He defined a context-aware system as “a system that uses context to provide relevant information and/or services to the user, where relevancy depends on the users task”. Another definition of context-awareness can be found in [Brown et al., 1997], where the authors defined a context-aware application as an application that changes its behavior according to the user’s present context.

In [Barkhuus and Dey, 2003], the authors described three levels of interactivity with respect to context awareness; personalized, active, and passive context awareness (Table 3.2). In personalized context awareness, users can customize or tailor the application to match their preferences. Both active and passive context awareness rely on sensory data. An active context-aware application can change its behavior autonomously while a passive context-
Table 3.2: Levels of interactivity in context-aware applications.

<table>
<thead>
<tr>
<th></th>
<th>Context Perceiving</th>
<th>Action Taking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personalized</td>
<td>User</td>
<td>User</td>
</tr>
<tr>
<td>Passive</td>
<td>Application</td>
<td>User</td>
</tr>
<tr>
<td>Active</td>
<td>Application</td>
<td>Application</td>
</tr>
</tbody>
</table>

A context-aware application presents the updated context to the user and lets the user decide how to change the application behavior [Chen et al., 2000].

For example, consider a lights control system that supports different lighting configurations. A personalized context-aware application will allow the user to select the lighting configuration of choice. In this case, the user perceives the context information and takes the action. A passive context-aware application will provide recommendations to the user when it detects a specific event (e.g., the user is reading). In this case, the application perceives the context information and the user takes the action. Finally, an active context-aware application will change the lighting configuration according to the user’s task. In this case, the application perceives the context information and takes the action autonomously.

Supporting context-aware interaction is crucial for SBEs. An SBE should support user tasks by presenting recommendations or taking autonomous actions. Users may feel less in control with active or passive context-aware interaction compared to personalized interaction. However, it has been shown that users are willing to give up partial control if they gain greater usefulness in return [Barkhuus and Dey, 2003]. Consequently, taking the appropriate decisions (i.e., recommendations and/or autonomous actions) based on the context information is a key factor in user’s acceptance of context-aware interaction. A context-aware application that often takes inappropriate decisions can be tedious and frustrating to the users.
3.4 User Interfaces

The ultimate goal of an SBE is to create a convenient ambiance for its users and to support their tasks efficiently. The incorporated things in an SBE can provide several functionalities that can facilitate user tasks. However, as the number of things increases, it becomes harder for the user to maintain a conceptual model that describes the different things and the relations between them. This can complicate the user experience rather than simplifying it. Ideally, an SBE should completely take over the control and meet all the needs of its users through autonomous actions. In fact, it has been shown that humans are willing to give up partial control and accept a large degree of autonomy as long as the reward in usefulness is greater than the cost of limited control [Barkhuus and Dey, 2003].

An SBE that can autonomously fulfill all the needs of its users without requiring them to do any cognitive effort does not need to support explicit user interaction. Relying solely on implicit user interaction eliminates the need for user interfaces. However, realizing such an SBE requires the provision of powerful tools that can infer all the factors affecting the experience of its users and address any conflicts between their needs, which is not yet practical. Consequently, autonomous actions have the potential to fail to meet user needs due to the absence of complete knowledge upon taking the decision. Moreover, a user might perceive autonomous actions as patronizing. Therefore, supporting explicit user interaction in IoT-enabled SBEs remains a must. Users should have the ability to intervene and override undesired autonomous actions. This requires an SBE to provide its users with some form of a user interface (UI) through which they can explicitly interact with the SBE.

A UI allows users to talk to the SBE (by providing input) and listen to it (by perceiving output). Several techniques for input and output have been proposed and many of them have well-established technologies. However, many of those techniques have originated in a world of desktop computers allowing for a very limited interaction that does not leverage the full interaction capabilities of human beings. In [O’Sullivan and Igoe, 2004], the authors have illustrated that by demonstrating how a desktop computer sees us. For a desktop
computer with a mouse, a keyboard, a monitor, and speakers, we look like a hand with one finger that provides input through sequential tapping, one eye to look at the two-dimensional monitor, and two ears to hear the stereo audio output. The human image perceived by a computer is reflected by its input and output capabilities. For a computer with a touchscreen (Figure 3.1a), we look like a poor creature with one eye to look at the two-dimensional screen and one finger to provide the touch input. For a smart speaker (Figure 3.1b), we look like a mouth that provides the voice commands and two ears to hear the stereo audio output or even a single ear if the audio output is mono. For a smartphone with a touchscreen, stereo audio output, and a microphone (Figure 3.1c), we look slightly better but still far from the full interaction capabilities of the human body.

Figure 3.1: How-a-computer-sees-us? [O’Sullivan and Igoe, 2004]

Relying solely on legacy input and output techniques to support user interaction in SBEs will place significant constraints on the user experience that an SBE can offer. It is crucial to adopt new input and output technologies that can help an SBE to better understand the actions and behavior of its users. This way, we can unleash interaction capabilities of the users allowing them to interact with the SBE as human beings rather than poor creatures. In explicit user interaction, a user may provide input through touch/press, speech, gesture, or thoughts. An SBE often provides output in the form of a Graphical User Interface (GUI), audio, or a Mixed-Reality (MR) user interface. A UI for an SBE may incorporate any combination of input and output techniques. A multi-modal UI incorporates several I/O techniques providing the users with multiple modes of interaction.
Attention is a scarce resource and many of us already suffer from information overload. When an application needs user input or has feedback, it typically engages the user immediately. As the number of things increases in an SBE, interrupting the user’s current task to request input or provide feedback becomes more often. Deviating the users from focusing on their current tasks can have a disruptive effect on both task performance and emotional state [Bailey et al., 2001]. A UI for an SBE should avoid unnecessary interrupts in order to help its users focus on their current tasks.

### 3.5 UI Usability

Traditionally, the usability of a UI has five associated attributes, which are learnability, efficiency, memorability, errors, and satisfaction [Nielsen, 1994]. Learnability refers to how easy the users can learn the user interface and rapidly use it to accomplish some work. Efficiency refers to how quickly the users can perform tasks once they have learned the user interface. Memorability refers to the ability of the users to remember the user interface and to reestablish proficiency after a period of not using it. Errors refer to how often the users make errors, the severity of these errors, and how easy it is to recover from them. Finally, satisfaction refers to how pleasant it is to use the user interface. The satisfaction attribute is highly dependent on the design specifics of the user interface as well as user preferences. Therefore, we will focus on the four former attributes when comparing the different types of input and output techniques that are commonly used in SBEs.

A comparison of different input techniques in terms of learnability, efficiency, memorability, and errors rate is shown in Table 3.3. Touch/press input is a well-established input technique that has been used by humans for decades. It is usually associated with a GUI or a physical control, which allows for providing the user with guidance or basic instructions. Benefiting from user exposure to previous similar systems, user interfaces relying on touch/press input can be designed to be learnable. In terms of efficiency, touch/press input requires the user to
Table 3.3: UI usability comparison for different input types.

<table>
<thead>
<tr>
<th>Input</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Learnability</td>
</tr>
<tr>
<td>Touch/Press</td>
<td>High</td>
</tr>
<tr>
<td>Speech</td>
<td>Low</td>
</tr>
<tr>
<td>Gesture</td>
<td>Low</td>
</tr>
<tr>
<td>Thoughts</td>
<td>High</td>
</tr>
</tbody>
</table>

have physical contact with the physical control or to navigate through a GUI to accomplish a given task, which often requires a considerable time and/or effort. Again, user familiarity with touch/press input can help improving memorability as well as minimizing errors. Both speech and gesture inputs allow users to perform tasks efficiently using simple inputs. However, they tend to have low learnability, where a user needs guidance to learn the correct voice commands and the specific gestures. Memorability for speech and gesture inputs tends to be low as well due to the fact that voice commands and gestures can vary dramatically between different systems and the difficulty of recalling them without guidance. Speech and gesture recognition technologies are less reliable compared to the well-established touch/press input technologies, which makes them more error-prone [Rowland et al., 2015]. Voice recognition and gesture recognition interfaces are intangible interfaces. Although sensing the surrounding space to infer input allows for hands-free interaction, it raises the possibility for false positive input. Thoughts recognition interfaces by definition require no learning nor memorability. Although it has the potential to perform tasks efficiently, accuracy remains a challenge given the current technologies.

Among the five senses, vision is the dominant one allowing a human being to perceive a lot of information at a glance. A comparison of different output techniques in terms of learnability, efficiency, memorability, and errors rate is shown in Table 3.4. GUI and MR interfaces are glanceable interfaces that target the sight sense of the user while audio interfaces target
Table 3.4: UI usability comparison for different output types.

<table>
<thead>
<tr>
<th>Output</th>
<th>Usability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Learnability</td>
</tr>
<tr>
<td>GUI</td>
<td>High</td>
</tr>
<tr>
<td>Audio</td>
<td>Low</td>
</tr>
<tr>
<td>Mixed-reality</td>
<td>High</td>
</tr>
</tbody>
</table>

The maturity of GUI design guidelines allows humans to benefit from their exposure to previous interfaces to learn new ones. Moreover, GUI and MR-based user interfaces can use visual elements to improve the learnability of the user interface. In contrast, users will probably need guidance to learn the meaning of different sounds provided through the audio output. For synthesized speech output, the language in use may act as a barrier preventing a user from perceiving the output. GUI tends to have low efficiency because it often requires manual navigation to capture different pieces of information. In contrast, mixed-reality interfaces allow a user to navigate to the thing of interest by simply gazing at it in the physical space. Compared to glanceable interfaces, audio interfaces provide output sequentially. Therefore, audio interfaces are efficient for short outputs but not for long ones. In terms of memorability, audio output requires the users to recall rather than recognize as the case for glanceable interfaces. The error rate associated with GUI and MR-based interfaces tend to be lower than that of audio interfaces. Due to factors such as noise, a user may fail to perceive an audio output correctly. In contrast, GUI and MR-based user interfaces allow users to perceive and verify information at a glance.

As SBEs gain popularity, the information around us will be overwhelming to the extent that we will crave a less distracting environment. Therefore, user interfaces should be non-intrusive and grab user’s attention only when required [Rowland et al., 2015]. Furthermore, the cognitive load associated with the UI should be kept as minimal as possible. Tangi-
ble/glanceable interfaces allow users to recognize the interface rather than completely recalling it from memory as the case for intangible/invisible interfaces. Therefore, the cognitive load for tangible/glanceable interfaces tends to be less than that of intangible/invisible ones. Users build their mental model of a system based on its user interface. For an SBE, that mental model should be consistent with the physical space. Otherwise, the UI can be confusing to the users. For instance, a GUI maps things in a 3-dimensional space into a 2-dimensional layout, requiring the user to develop a mental model that is different from the physical one. Such mapping can make interaction tricky to the user, especially if the GUI does not adapt to the user position and orientation.

Nowadays, the two most common forms of user interfaces for things are smartphone applications and web-based portals. Both rely on GUI and touch/press to provide input. Perhaps, the adoption of GUI is driven by the availability and maturity of well-established GUI-based interaction techniques and technologies. However, it should be taken into consideration that GUI interfaces were actually developed to interact with the digital world and adapting them to support interaction with the physical world may not always be straightforward. In the digital world, developers have the freedom to decide how to construct and present that world to the users and consequently what interaction techniques and user interfaces to support. On the other hand, interaction with things should consider the peculiarities and constraints associated with interaction in the physical world. In fact, supporting interaction in the physical space should be a motive to explore new interaction techniques and novel user interfaces.

With recent advances in MR technologies, MR devices are becoming more affordable and popular. This provides a great opportunity to use MR headsets as interaction proxies in SBEs. MR can help enriching the physical space with virtual objects through which the user can interact with things. Unlike GUI, MR interfaces allow for a straightforward mapping between the UI and the physical space. Virtual objects can be perfectly aligned with different things in the physical space by leveraging the spatial mapping capability of the MR device. However, requiring the users to use a head-mounted device to interact with SBEs might not be always convenient. Future advanced in MR technologies may provide affordable space-
mounted devices that are able to project virtual objects into the physical space without requiring the user to use wearable devices.

Usability is an important factor when deciding which input and output techniques to incorporate in a UI for an SBE. However, there are other factors that may affect that decision such as the availability of the required capabilities to support a given input or output technique. Accomplishing a given task in an SBE requires the user to follow some interaction scenario. An interaction scenario consists of a list of interactions, each may favor one input or output technique over another. Moreover, the current context and user preferences may play a significant role in deciding which input or output technique is more convenient to perform a given interaction. Therefore, an SBE UI may support a variety of input and output techniques to support different interaction scenarios and to provide the users with the flexibility of multi-modal interaction.
Part II

Problem and Approach
Problem Definition

An SBE, due to its intelligence, is a physical space that can autonomously acquire and apply knowledge about its surroundings and adapt to its users in order to improve their experience of the space and make their lives more comfortable [Cook and Das, 2004]. As a CPS, it uses the IoT technologies as a foundation to realize its capabilities and benefits from Internet access to leverage cloud services and to collect information from several data sources besides its sensors such as social networks, streaming services, online calendars, etc. Applying knowledge may take the form of autonomous actions or recommendations presented to the user. Besides being intelligent, an SBE accommodates for other aspects including control, enterprise, and material and construction [Buckman et al., 2014]. Accommodating for control requires an SBE to support explicit user interaction, allowing its users to interfere and override inappropriate autonomous actions if any. This chapter explains the challenges associated with supporting explicit user interaction in SBEs within a defined scope.
4.1 Problem Scope

The IoT involves connecting physical objects (or things) to the Internet. SBEs are one of the several applications of the IoT. Legacy SBEs were mostly closed systems that are accessible only locally and operate using their own standards and protocols. Consequently, their capabilities and provided functionalities were limited to the available local resources. This research focuses on IoT-enabled SBEs, where an SBE is a CPS that is connected to the Internet and exploits the various IoT technologies to realize its capabilities. Being connected to the Internet, an SBE can leverage the cloud resources and the IoT services to provide enhanced functionalities that were not applicable (or feasible) in closed systems.

As mentioned in section 3.2, user interaction can be categorized as either implicit or explicit. Implicit user interaction refers to the spontaneous interaction that does not involve cognitive effort at the user side but rather relies on context information to take autonomous actions on behalf of the user. Ideally, an SBE should act autonomously to meet the needs of its users and address conflicts if any. However, relying solely on implicit user interaction to meet user needs is not always applicable given the current technologies. Consequently, an SBE should continue to support explicit user interaction as well, allowing users to be in control and to override inappropriate autonomous actions if any. This research is mainly concerned with supporting explicit user interaction in SBEs, where an SBE should provide its users with multimodal UIs through which they can cognitively provide input and/or receive output.

The scale of an SBE can range widely, from the smallest usable physical space to the concept of a smart planet. This research focuses on SBEs at the room level, where a room acts as an SBE unit. A smart room is an indoor contiguous physical space with incorporated things that are connected to the Internet. We consider a smart building as a collection of SBE units (or smart rooms).
4.2 Interaction Proxy

Traditional computing devices are usually equipped with the necessary Input/Output (I/O) capabilities to provide a fully-functional UI that allows a user to interact with the device directly and use all of its functionalities. In contrast, things come in a wide variety of form factors with varying I/O capabilities. A thing might have limited I/O capabilities that it cannot provide a fully functional UI on its own. In such case, thing vendors usually rely on smartphone apps and/or web-based portals to provide a UI through which a user can access all the functionalities provided by their products. Consequently, unlike traditional computing devices, a user does not interact with a thing directly but rather through a third-party entity that provides the UI on behalf of that thing. We will be referring to that third-party entity as Interaction Proxy.

An interaction proxy is a single computing device (or a set of cooperating devices) with communication and I/O capabilities that acts as an intermediary between users and things by providing the UI through which they can interact.

The use of interaction proxies is not limited to support user interaction with things that lack built-in UIs. Although a thing might have a fully functional built-in UI, the use of an interaction proxy might still be feasible. For instance, an interaction proxy can provide access to remote things whose built-in UIs are not reachable at the moment. Furthermore, a thing might provide a simple built-in UI and rely on an interaction proxy to provide a more advanced UI with more functionalities or to support other interaction modals.

An interaction proxy may take several forms, which can be broken into four dimensions (complexity, purpose, location, and exposure) as shown in Figure 4.1. In terms of complexity, an interaction proxy may range from a single device providing a simple UI to a network of devices operating synergically to provide a multimodal UI. In terms of purpose, an interaction proxy can take the form of a general-purpose computing device or a special purpose device. Given the popularity of smartphones and other general-purpose computing devices, thing
vendors often provide UIs for their products in the form of smartphone apps and/or web-based portals. In contrast, an interaction proxy may take the form of a special-purpose device dedicated only to provide a UI for a specific thing type. For instance, the Philips Hue Tap, shown in Figure 4.2, is a special-purpose device that acts as an interaction proxy between users and Phillips lights by providing a UI consisting of four programmable physical buttons. In terms of location, an interaction proxy can be a user-side one such as handheld and wearable computing devices or it can be an SBE-side one that is embedded into the SBE. A multiple-device interaction proxy might have a mixture of user-side and SBE-side devices as well as general-purpose and special-purpose devices. Finally, an interaction proxy can be hidden or exposed to the user. A hidden interaction proxy works behind the scenes to support user interaction with things without exposing its physical embodiment to the user. On the other hand, exposed interaction proxies act as explicit gates between users and things. user-side interaction proxies are often exposed ones (e.g. wearable devices and smartphones) while SBE-side interaction proxies can be either hidden or exposed.

![Diagram of interaction proxy dimensions](image)

Figure 4.1: The four form dimensions of an interaction proxy.

Different interaction proxies have varying form factors and I/O capabilities. This can place a restriction on the UI types they can support. It is important to understand the relation between the interaction proxy form and the I/O types it can provide. Thereby, a UI designer
can determine which interaction proxy form to use in order to support the required I/O capabilities for a given UI type. Knowing the form of the interaction proxy can give an initial idea about the required SBE setup and the expected user experience.

Touch/press, speech, gesture, and thoughts are four input types commonly used in UIs for SBEs. Table 4.1 shows the interaction proxy forms that can support each of these input types. Tough/press input requires physical contact with the interaction proxy. Therefore, an interaction proxy for touch/press input must be exposed to the user whether it is a user-side one (e.g. a smartphone) or an SBE-side one (e.g. an SBE-embedded device with a touchscreen or physical controls). For speech input, an interaction proxy should allow users to provide inputs via voice commands. Such proxy can be a user-side exposed one (e.g. a smartphone) or an SBE-side one whether exposed or hidden (e.g. an SBE-embedded device with a microphone and speech recognition capability). For gesture input, an interaction proxy can be a user-side exposed one (e.g. a wired glove) or an SBE-side one whether exposed or hidden (e.g. an SBE-embedded device with a camera and gesture recognition capability). For thoughts input, a brain-computer interface is commonly used, which is considered an exposed user-side interaction proxy.
Table 4.1: Input type support in different interaction proxy forms.

<table>
<thead>
<tr>
<th>Input type</th>
<th>Interaction proxy form</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exposure</td>
<td>Location</td>
</tr>
<tr>
<td></td>
<td>Hidden</td>
<td>User-side</td>
</tr>
<tr>
<td>Touch/Press</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Speech</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Gesture</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Thoughts</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Graphical User Interface (GUI), audio, and Mixed-Reality (MR) are three output types commonly used in UIs for SBEs. Table 4.2 shows the interaction proxy forms that can support each of these input types. A GUI is often presented through a screen, which must be visible to the user. Therefore, the interaction proxy must be exposed. In terms of location, the interaction proxy can be a user-side one (e.g. a smartphone with a built-in screen) or an SBE-side one (e.g. a screen mounted in the SBE). Audio playback can take place through exposed user-side interaction proxies (e.g. smartphones) or SBE-side interaction proxies (e.g. speakers), whether exposed or hidden. For MR output, an MR headset is commonly used, which is considered an exposed user-side interaction proxy. Future advances in MR technologies may provide affordable MR devices that are SBE-mounted rather than head-mounted. Such SBE-side devices can be either exposed or hidden from the user.

Besides the benefits of using interaction proxies to support user interaction with things, there are consequences that can be of concern. An interaction proxy may introduce a barrier between users and things. Relying solely on interaction proxies to support explicit user interaction with things, makes the availability of these interaction proxies essential for the interaction to take place. Consequently, lack of access to interaction proxies can make
Table 4.2: Output types support in different interaction proxy forms.

<table>
<thead>
<tr>
<th>Output type</th>
<th>Interaction proxy form</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Exposure</td>
<td>Location</td>
</tr>
<tr>
<td></td>
<td>Hidden</td>
<td>Exposed</td>
</tr>
<tr>
<td>GUI</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Audio</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>MR</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

things inaccessible to the users. Moreover, multiple user may interact with the same thing simultaneously using several UIs hosted on different interaction proxies. This requires an SBE to support conflict resolution and to provide mechanisms to maintain consistency between the current state of its incorporated things and all of their corresponding UIs hosted on different interaction proxies.

A hidden interaction proxy should blend into the SBE and work behind the scenes to support a transparent interaction experience between users and things. For instance, a user may interact with a given thing using gestures without being aware of the interaction proxy that captures and interprets those gestures before communicating them to the intended thing as machine-readable commands. On the other hand, an exposed interaction proxy is visible to the user and acts as an explicit gate between users and things. For instance, a user may interact with a given thing using a smartphone, which requires the user to hold the device and use a specific application. In general, the use of exposed interaction proxies should be avoided whenever possible in favor of a seamless transparent interaction between users and things. Although the use of user-side interaction proxies such as smartphones and other wearable devices can help provide the user with a personalized interaction experience, the responsibility of maintaining and configuring those devices usually rely on the user. On the
other hand, SBE-side interaction proxies are usually preconfigured to support interaction with the SBE they are embedded in. Thereby, users are relieved of the responsibility of maintaining or configuring them.

4.3 Interaction Challenges

An IoT-enabled SBE can incorporate numerous things and have several users, which requires the orchestration of heterogeneous things with varying capabilities and standards as well as fulfilling the needs of users with different preferences and goals. The fact that thing UIs are often provided through interaction proxies, as discussed in section 4.2, adds several interaction complexities that we did not experience in interacting with traditional devices with built-in UIs. In order to support explicit user interaction in IoT-enabled SBEs, several challenges need to be addressed. The following subsections illustrate some of those challenges.

4.3.1 Consistency

A thing may have a built-in UI as well as several other UIs provided by different interaction proxies. It is important to keep the state of all UIs for a thing consistent and ensure that they all reflect the current state of that thing. Users should be able to switch seamlessly between different thing UIs and still experience a fluent interaction, where they can begin a task using one UI and resume it using another. In order to maintain consistency between a set of UIs, a change in one UI should quickly propagate to the other UIs. This can be challenging due to network glitches that can cause the UIs to go out of sync. Maintaining consistency between all things and UIs in an SBE can introduce a significant overhead and cause performance issues, especially as the number of things and UIs increase. This requires careful design and selection of scalable synchronization techniques.
4.3.2 Interoperability

Things come in various form factors and I/O capabilities. Usually, things provided by the same vendor utilize compatible technologies and can interoperate. However, due to the lack of IoT standardization, as mentioned in section 3.1, it is usually challenging to achieve interoperability between things provided by different vendors. Recalling that vendors usually provide thing UIs in the form of smartphone apps or web-based portals, some vendors provide smartphone apps that can support several thing types (e.g. Samsung SmartThings, Belkin WeMo) rather than providing a different smartphone app per thing type. However, such apps tend to support things provided by specific manufacturers. Yet, there is a lack of smartphone apps that can globally support interaction with things regardless of their type or manufacturer.

4.3.3 Inter-Usability

Different UIs for different thing types provided by different vendors tend to have varying designs. It is important to consider not only the usability of distinct thing UIs but also their inter-usability [Denis and Karsenty, 2004]. An SBE should provide its users with coherent thing UIs rather than disjoint heterogeneous UIs. Coherent UIs should have consistent conceptual models, functionality organization, and interaction logic as well as consistent terminologies and visual designs if applicable. This can be challenging in an SBE that incorporates things provided by different vendors. Achieving coherency between UIs provided by different vendors requires those vendors to agree on using specific UI design standards, which is not usually achievable in industry practice.

4.3.4 Distributed Functionality

In traditional isolated devices, there is a many-to-one relationship between functionalities and devices, respectively, where a device can provide multiple functionalities and a function-
ality is fully provided by a single device. In contrast, there is a many-to-many relationship between functionalities and things in an SBE, where a functionality can be distributed across multiple things and a thing can be utilized by different services to provide multiple functionalities. Figure 4.3 shows a functionality distribution comparison between traditional isolated devices and SBE things in a typical house setup. In a traditional house, devices are isolated from each other, where each device provides certain functionalities. Using a given functionality (e.g. cooking), requires the user to interact with a single device (e.g. oven). In contrast, a smart house provides its users with several services, where each service can utilize one or more things to provide a set of functionalities. For example, a lighting service can utilize multiple light bulbs to provide illumination and motion sensors to control the lights based on user proximity. On the other hand, a thing can be utilized by one or more services. For instance, a light bulb can be utilized by a lighting service to provide convenient illumination, a security service to simulate user presence, and a power saving service to reduce power consumption.

Figure 4.3: Functionality distribution across things in an SBE compared to traditional isolated devices in a typical house setup.

The straightforward mapping between functionalities and devices in traditional built environments allowed the users to easily identify which device to interact with based on the required functionality. Therefore, users of traditional built environments used to interact at
the device-level. In contrast, interacting with an SBE at the thing-level may not always be feasible. As the number of things increases in an SBE, it becomes more challenging for the user to control them individually. Moreover, as the number of supported services increases, the interrelationships between things and services become more complex. Consequently, it becomes challenging for the users to understand those relationships and to develop a conceptual model of which thing does what, when, and why. Compared to traditional built environments that provide device-based functionalities, an SBE tends to provide service-based functionalities as well. Therefore, besides supporting user interaction at the thing level through thing UIs, an SBE should support user interaction at the service level.

4.3.5 Conflict Resolution

An SBE can have multiple users with different needs. A need refers to anything we value highly, whether it is a tangible deficiency (e.g. “I need lights”) or intangible aspiration or goal (e.g. “I need peace of mind”) [Rowland et al., 2015]. Needs have varying priorities, where they can be of minor value or critical to survival. Users of an SBE may have conflicting needs. Therefore, an SBE should support conflict resolution. Sometimes, it is impossible to fulfill the needs of all users or reach a compromise. In such case, an SBE might pick a side based on predefined criteria.

A user need may also conflict with a service goal. For example, a power saving service may decide to reduce the intensity of lights to save energy while the user prefers to keep the lights at high intensity. In such case, the SBE should enable the user to understand the source of the conflict and to override the service behavior if applicable.

4.3.6 Usability

The availability of mature, popular, and well-established HCI technologies that have evolved over decades may have encouraged SBE designers and thing vendors to adopt those tech-
nologies to support user interaction. However, those legacy technologies place several limits on the interaction as discussed in section 3.4. In fact, those technologies were originally developed to support user interaction with the digital world. However, interacting with the physical world is a different context that will require adapting the current interaction technologies or adopting new ones.

As mention in section 4.2, an interaction proxy can be a special-purpose or a general-purpose one. A special-purpose interaction proxy provides a UI for a specific thing type while a general-purpose interaction proxy can provide UIs for several thing types. For special-purpose interaction proxy, the user will need to use the correct interaction proxy in order to interact with a given thing. Similarly, for general-purpose interaction proxies, the user will need to use the correct UI before interacting with the thing of interest. As the number of things increases in an SBE, the number of UIs that the user will need to select from will increase as well. This can have implications on the usability of the SBE as follows:

- **Learnability.** Before users can interact with a given SBE, they need to be aware of what capabilities does it provide and how to use those capabilities. Expecting users to have such knowledge is not always a valid assumption. For instance, non-domestic users (e.g. visitors) of an SBE tend to be unaware of its capabilities nor the UIs it provides. Therefore, unless those users receive some kind of guidance, it can be tricky for them to learn about the SBE or even infeasible, especially during a short visit. Moreover, the UIs provided by different thing vendors tend to utilize varying designs and technologies. Similarly, SBEs with different configurations tend to have different UIs with varying designs and interaction techniques. Learning numerous and varying UIs to interact with different things and SBEs can be overwhelming to the users or at least will involve a high learning curve.

- **Efficiency.** An interaction proxy provides users with a UI through which they can interact with things. Meanwhile, it can introduce a barrier between users and things. Lack of access to interaction proxies makes things inaccessible to the users. Therefore,
the presence of the interaction proxy is essential for the interaction to take place. Relying on a third-party entity to support interaction between humans and things may not always be feasible as it may require more interactions to accomplish simple tasks. For example, turning the lights on/off using a smartphone application can involve more interactions compared to the use of a traditional physical light switch. Moreover, requiring the user to manually switch between different UIs to interact with different things/SBEs can add a significant overhead by increasing the number of interactions required to accomplish a given task. An SBE should help improve the efficiency of its users in accomplishing their tasks. Failing to support efficient user interaction compared to traditional built environments violates an essential goal of SBEs.

**Memorability** Interacting with different things and/or SBEs requires the user to switch between different UIs. As the number of things/SBEs increases around us, the number of corresponding UIs will increase as well. Having numerous UIs with inconsistent designs can be overwhelming to the user, making it harder to memorize them and to rapidly re-establish proficiency after a period of not using them.

**Errors.** As the number of UIs increase dramatically, a user aiming to accomplish a given task will be more subject to select the wrong UIs before figuring out the correct one. Moreover, having several inconsistent UIs can confuse the user and thereby cause more errors. Furthermore, recovering from errors in an SBE can be challenging due to the complex interrelationships between things and services that makes it tricky for the users to figure out the reasons behind a given error.

**Satisfaction.** Manually switching between different UIs to accomplish different tasks can be frustrating and tedious to the user, especially as the number of those UIs increase. An SBE that is tricky to learn, hard to memorize, inefficient to use, and error-prone has less potential to achieve user satisfaction.
4.3.7 Context Awareness

As mentioned in section 3.2, user interaction in an SBE can be either implicit or explicit. Supporting context-awareness in implicit user interaction focuses on leveraging context information to enable things to take autonomous actions. On the other hand, supporting context awareness in explicit user interaction focuses on leveraging context information to adapt the UI. A context-aware UI should facilitate user tasks by autonomously adapting to the current context, thereby reducing the required interactions. However, failing to infer the current context correctly can result in invalid UI adaptations that will require manual user intervention to correct them. A UI that performs incorrect adaptations frequently can be tedious and frustrating to the users. Designing context-aware UIs is a challenging task. It is a double-sided weapon, where its success can enhance the user experience significantly but its failure can degrade the user experience dramatically.

4.4 Research Objectives

While there has been a considerable research effort to address a variety of challenges associated with supporting thing-to-thing interaction, human-to-thing interaction related research is limited. This research is concerned with human-to-thing interaction and focuses on supporting explicit user interaction in IoT-enabled SBEs. In section 4.3, we have explored several interaction challenges that an SBE needs to address. The main objective of this research is to

“Support explicit user interaction in IoT-enabled SBEs by providing multimodal, consistent, coherent, and adaptive UIs while reducing the physical and mental efforts that are required at the user side to obtain and use those UIs.”

Multimodal UIs allows an SBE user to leverage the full capabilities of the human-body to interact and engage with the SBE. Consistent UIs allow for a fluent interaction experience,
where a user can begin a task using one UI and resume it using another. Coherent UIs can help achieve inter-usability, which can improve the learnability and memorability of those UIs and make them less confusing the user and thereby less error prone. Adaptive UIs can facilitate user tasks by reducing the required interactions to accomplish those tasks.

As the number of things and SBEs around us continues to increase it tends to be impractical for users to obtain and use numerous custom vendor-provided UI implementations to control hundreds or thousands of individual things. Assisting users by minimizing the physical and mental efforts that are required to obtain the appropriate UI for a given task in a given SBE has the potential to both increase user performance and reduce the mental workload.

Relying on SBE-side interaction proxies can help minimizing the overhead at the user side. Unlike most user-side interaction proxies, SBE-side ones are often preconfigured and ready for instant use. However, this requires the SBE to incorporate additional equipment, which can increase its implementation and maintenance cost. On the other hand, user-side proxies can offer a cost-efficient solution as they are often mobile and reusable, allowing a user to interact with different SBEs using the same interaction proxy. Moreover, SBE-side interaction proxies do not support remote interaction as they require the user to be present within a defined scope in order to provide input and perceive output.

A user-side interaction proxy usually takes the form of a general-purpose personal computing device (e.g. a smartphone). Unfortunately, users are often required to manually configure their interaction proxies by installing thing-specific or SBE-specific UI applications. Moreover, such approach requires the users to explicitly switch between several applications to interact with different things/SBEs. This requires a user to perform additional interactions and introduces a significant mental workload. Although web portals avoid the need for software installations, users are still required to manually switch between different web-portals to interact with different things. Ideally, an interaction proxy should be able to provide the UI for a given SBE autonomously. Thereby, the interaction overhead at the user side can be minimized. We formulate our research goal to be.
"Enable an interaction proxy to autonomously discover and learn about the available SBEs, the things they incorporate, and the services they provide before generating an adaptive and always up-to-date UI given the available I/O capabilities."

Achieving this goal has the potential to improve user interaction with SBEs in several ways. First, supporting user interaction with SBEs using a variety of interaction proxy types with varying I/O capabilities allows for multimodal interaction (i.e. users can leverage the full interaction capabilities of the human body). Second, enabling the interaction proxies to autonomously discover and learn about the available SBEs exempts the users from being required to have prior knowledge about the available SBEs and their capabilities before interacting with them (i.e. non-domestic users can interact with an SBE without guidance). Third, learning about not only the incorporated things but also the provided services allows an interaction proxy to support user interaction at both the thing level and the service level (i.e. users will not be limited to interact with things individually). Forth, generating the UIs for things using the same design pattern can help achieving coherency between them (i.e. inter-usability across different UIs can be achieved). Fifth, adaptive UIs can autonomously adapt to the current context and the task at hand (i.e. the required user interactions to accomplish a given task can be reduced). Finally, UIs that are always up-to-date will always be consistent and reflecting the current state of the corresponding things and/or services (i.e. a user can begin a task using a UI and resume that task using another).

In terms of SBE usability, realizing our research goal provide several rewards. First, it allows new users to quickly learn about an SBE and instantly start interacting with it (i.e. it can improve the learnability of the SBE). Second, it allows returning users to recognize the automatically provided UI rather than manually recalling the thing-specific or SBE-specific UI (i.e. it can improve the memorability of the SBE). Third, the UI can autonomously adapt to the current context to facilitate user tasks (i.e. it can improve the efficiency of user interaction). Forth, as the number of things and the corresponding UIs increases, users will be more subject to select the wrong UI before figuring out the correct one. Relieving users
from the task of manually selecting the UI for each task can help avoiding such common mistakes (i.e. it can reduce user errors). Finally, minimizing the required physical and mental workload at the user side can help reducing user frustration and tediousness (i.e. it can improve user satisfaction).

Fulfilling this research goal requires addressing the following research questions.

- **How to discover an SBE?** Before supporting user interaction with an SBE, an interaction proxy should first be aware of its existence. An interaction proxy should have the ability to discover the available SBEs whether in user surroundings or at a remote location.

- **How to learn about an SBE?** An interaction proxy should be able to inquire the SBE about its incorporated things and provided services as well as their current state and supported functionalities. In response, the SBE should have the ability to provide interpretable answers.

- **How to generate the UI based on the available I/O capabilities?** Different types of interaction proxies tend to have varying I/O capabilities. Consequently, the UI types that they can support tend to vary as well. Therefore, the SBE should not place any assumptions about the capabilities of the interaction proxies when replying to their queries. Instead, the SBE response should include abstract information that is independent of any technology, capability, or platform. Using that abstract response, an interaction proxy regardless of its type should be able to generate a UI that leverages its I/O capabilities.

- **How to benefit from context information to adapt the UI?** Although an SBE may be crowded with things and services, its UI should not reflect that to the user. Overwhelming the user with features and functionalities that are not necessarily of interest at the moment can complicate the user experience. The interaction proxy should leverage context information to adapt the UI accordingly. The UI may prioritize
certain things over others based on the task at hand or perform automatic navigation based on user location. Context information can be obtained from the SBE and/or the interaction proxy itself. In either case, this information should be applied in order to provide the user with a customized UI that can facilitate the current user task. Adapting the UI is an autonomous decision that should be taken with care to ensure user satisfaction. A wrong decision can confuse the user and degrade the interaction experience. Moreover, overriding a wrong decision introduces unnecessary additional interactions that can be tedious to users.

- **How to maintain consistency between an SBE and its UIs?** Things and their UIs are often decoupled, where several UIs for the same thing can be hosted on different interaction proxies. Communication between things and their UIs takes place via exchanging messages over a communication network. Due to network glitches, maintaining consistency between things and their UIs can be challenging. Moreover, it can affect the responsiveness of the SBE to user actions, which can dramatically degrade the user experience. Therefore, it is crucial to ensure reliable and responsive communication between things and their UIs.

The research questions mentioned above are addressed in [chapter 5](#) that presents a framework to addresses the first four research questions of how to discover and learn about an SBE before generating and adapting its UI and provides before explaining how to address the research question concerned with maintaining consistency between an SBE and its UIs.
Chapter 5

Proposed Approach

Explicit user interaction in SBEs usually takes place through custom vendor-provided apps. This requires the user to download and use several apps to interact with different things and services. Such an approach does not scale well as the number of things and services continues to increase. Switching between numerous apps to interact with different things and services can introduce a significant mental workload and decrease user performance. Moreover, SBEs are usually dynamic, where things are added/removed and services are created/destroyed over time. Consequently, users will need to perform frequent software updates to catch up with changes in SBEs, thus increasing the required efforts on users.

Besides the drawbacks mentioned above, several challenges that are associated with supporting explicit user interaction in SBEs were discussed in section 4.3 before identifying the main research objective along with a set of research questions in section 4.4. Enabling the interaction proxies to automatically provide the required UIs for different SBEs can help address many of the interaction challenges that SBE users often face with the currently adopted approaches. In order to provide SBE UIs automatically, four main research questions need to be addressed, which are how to (1) discover and (2) learn about an SBE before (3) generating and (4) adapting its UI. Unlike most of the traditional discovery services, SBE discovery should consider location information and allow for remote discovery. Learning should allow
for acquiring enough information about things and services in an SBE before generating the UI. UI generation should not be focused on GUIs as the case for most of the traditional UI generation approaches. This chapter presents a solution approach to enable an interaction proxy to autonomously provide the necessary UIs to support explicit user interaction with SBEs. The solution approach is not restricted to a specific UI type but rather considers the varying I/O capabilities of different interaction proxies.

Before moving forward with explaining the solution approach, it is important to understand how explicit user interaction may take place in an SBE. This requires identifying the entities involved in the interaction and clarifying the relationships between them. Figure 5.1 shows a proposed framework to describe explicit user interaction in SBEs. The framework leverages the Interaction Proxy concept that was coined in section 4.2. It describes an SBE as a collection of things, services, and SBE-side interaction proxies. A service may utilize one or more things to provide a set of functionalities. Both user-side and SBE-side interaction proxies allow users to interact with the SBE at the service-level and/or the thing-level. In addition, users may interact directly with things using their built-in UIs if available.

Figure 5.1: A framework to describe explicit user interaction in SBEs.
Supporting direct user interaction using thing built-in UIs is mostly dependent on the form factors and the UI designs selected by thing vendors, which is beyond the scope of this research. This work focuses on supporting explicit user interaction in SBEs using interaction proxies. An interaction proxy should provide the UI through which a user can interact with things/services in an SBE. An SBE-side interaction proxy is often tightly coupled with its corresponding SBE, where it is usually preconfigured for that SBE and ready for instant use. However, providing SBE-side interaction proxies often requires incorporating additional equipment in the SBE. In contrast, user-side interaction proxies allow for reusability, where the same interaction proxy (e.g. a smartphone) can support user interaction with different SBEs. However, a user-side interaction proxy needs to be configured for each SBE before it can support user interaction with it.

For an interaction proxy to automatically provide the UI for a given SBE, it should be able to discover and learn about that SBE before generating the UI and adapting it to the current context. Figure 5.2 shows a four-step process that an interaction proxy may follow to automatically generate a UI for a given SBE. The first step is to discover the available SBEs before selecting the SBE of interest. The second step is to obtain information about the incorporated things and the provided services of the selected SBE. The third step is to use the obtained information to generate a UI based on the I/O capabilities of the interaction proxy. Finally, the interaction proxy should benefit from context information to adapt the UI in order to facilitate user tasks.

![Figure 5.2: A four-step process to automatically generate a UI for an SBE.](image)
Automatic generation of an SBE UI requires three basic ingredients, which are thing/service information, I/O capabilities, and context information. thing/Service information refers to information about the incorporated things and the provided services of the SBE. This information should be available in a machine-readable format and use a common unified language that can be interpreted by any interaction proxy regardless of its capabilities or the technology it uses. The interaction proxy should be aware of its I/O capabilities, which determine the UI type(s) it can provide. Context information can help adapt the UI to enhance the interaction experience. The interaction proxy can obtain context information from various sources either directly using its own capabilities and/or indirectly using the capabilities of the SBE.

Realizing the four-step process depicted in Figure 5.2 requires an infrastructure that offers a set of functionalities, which an interaction proxy can utilize to provide the UI automatically. Figure 5.3 shows a proposed framework for automatic generation of SBE UIs. The framework consists of two main parts (the infrastructure and the interaction proxy). The infrastructure part consists of an SBE ontology instance in addition to three services for discovery, learning, and registration. The interaction proxy part consists of four modules for discovery, Learning, UI generation, and UI adaptation. Several interaction proxies may utilize the same infrastructure.

The SBE ontology instance, as explained later in subsection 5.1.1, stores detailed information about things and services together with the SBEs incorporating them, especially the information that is relevant to UI generation (e.g. location, functionalities, etc). The discovery service uses the information stored in the ontology to respond to SBE discovery requests sent by the interaction proxies. The Learning service provides machine-readable thing/service descriptions that are extracted from the SBE ontology instance. This description can be interpreted by any interaction proxy regardless of its type or the technology it uses. The registration service allows for registering new things/services by collecting information about them before storing it in the SBE ontology instance.
The discovery agent allows an interaction proxy to discover the available SBEs by sending a discovery request to the discovery service. The discovery agent may use context information (e.g., user location) to customize the discovery request in order to discover SBEs that meet a specific criteria. The learning agent obtains thing/service descriptions for the designated SBE(s) by sending a request to the learning service that specifies the SBE(s) of interest. The UI generator module interprets the thing/service descriptions obtained by the learning agent to learn about the attributes and functionalities of the things/services before generating the UI based on the available I/O capabilities. Finally, the UI adapter module adapts the generated UI based on the available context information. UI adaptation is a continuous process that takes place over the life-cycle of the generated UI.

The following sections will cover in detail the different aspects of the proposed automatic UI generation framework. Section 5.1 describes an interoperable approach to store thing/service
information and explains a methodology to automate thing/service registration. \[\text{section 5.2}\] and \[\text{section 5.3}\] cover the discovery service and the learning service, respectively. UI generation is discussed in \[\text{section 5.4}\] and UI adaptation is illustrated in \[\text{section 5.5}\].

## 5.1 Thing/Service Information

Things and services may utilize different technologies and standards to realize their capabilities and supported functionalities. Referring to the automatic UI generation framework depicted in Figure 5.3, the thing/service information repository should store information about the available things and provided services. Storing this information in a coherent format can be challenging due to heterogeneity concerns, especially between things provided by different vendors. This can be addressed by representing thing/service information using a semantic-based approach. An ontology-based approach to store thing/service information is presented in \[\text{subsection 5.1.1}\]. Thing/service registration can take place through the registration service, which is illustrated in \[\text{subsection 5.1.2}\].

### 5.1.1 SBE Ontology

This subsection describes a proposed ontology to represent thing/service information for IoT-enabled SBEs, namely SBE ontology. The proposed SBE ontology is an extension of the IoT-Lite ontology. The IoT-Lite ontology [Bermudez-Edo et al., 2017], as mentioned in \[\text{subsection 2.3.1}\], is a lightweight instantiation of the widely known Semantic Sensor Network (SSN) ontology [Compton et al., 2012] to describe IoT resources, entities, and services while remaining suitable for constrained IoT environments. However, it has limited support for location-aware applications in indoor settings.

Location-awareness is crucial for IoT-enabled SBEs, where the location information can be used to provide adequate services or to adapt the UI. The IoT-Lite ontology supports
location-awareness through the *geo:Point* concept (Figure 2.1) that allows for specifying both quantitative and qualitative positioning information. The *geo:Point* concept has three data-type properties for specifying quantitative positioning information (*geo:lat*, *geo:long*, and *geo:alt*) and two data-type properties for specifying qualitative positioning information (*iot-lite:relativeLocation* and *iot-lite:altRelative*). The *geo:lat*, *geo:long*, and *geo:alt* properties can be used to specify latitude, longitude, and altitude values, respectively. The *iot-lite:relativeLocation* property can be used to specify a qualitative location (e.g. building A, New York) while the *iot-lite:altRelative* property can be used to specify a qualitative elevation (e.g. floor 2).

Table 5.1: IoT-Lite ontology support for positioning information.

<table>
<thead>
<tr>
<th></th>
<th>Quantitative</th>
<th>Qualitative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indoor</td>
<td>No support</td>
<td><em>iot-lite:relativeLocation</em> and <em>iot-lite:altRelative</em></td>
</tr>
<tr>
<td>Outdoor</td>
<td><em>geo:lat</em>, <em>geo:long</em>, and <em>geo:alt</em></td>
<td><em>iot-lite:relativeLocation</em> and <em>iot-lite:altRelative</em></td>
</tr>
</tbody>
</table>

The IoT-Lite ontology support for location-aware indoor applications suffers from two main limitations. First, it relies on GPS data to specify quantitative positioning information. However, the use of GPS is not practical for indoor applications because common GPS receivers do not work indoor. Second, the use of *iot-lite:relativeLocation* to specify qualitative positioning information can lead to ambiguity, as it can refer to a room, a building, a city, etc. The proposed SBE ontology extends the IoT-Lite ontology to address those limitation by providing additional concepts and properties to support location-awareness in indoor settings using both quantitative and qualitative positioning information.

The SBE ontology, as shown in Figure 5.4, provides four new concepts and four new object properties. The *sbe:Building* and *sbe:Space* concepts provide unambiguous support for qualitative indoor positioning information. The *sbe:Transform* and *sbe:Origin* concepts provide support for quantitative indoor positioning information, which is missing in the IoT-Lite ontology. The *sbe:contains* object property can assign an *sbe:Space* object to an *sbe:Building*.
object. The `sbe:hosts` object property can assign an `ssn:platform` object to an `sbe:Space` object. The `sbe:hasTransform` object property can assign an `sbe:Transform` object to an `ssn:Platform`, `ssn:Device`, or `sbe:Origin` object. Finally, the `sbe:hasOrigin` object property can assign an `sbe:Origin` object to an `sbe:Transform` object.

In addition to the new concepts and object properties, the SBE ontology provides new data-type properties. As shown in Table 5.2, the `sbe:Building` concept has one data-type property, the `sbe:Space` concept has three, and the `sbe:Transform` concept has nine. The `sbe:Origin` concept has no associated data-type properties. The `sbe:positionX`, `sbe:positionY`, and `sbe:positionZ` data-type properties allows for specifying position information in meters. The `sbe:rotationX`, `sbe:rotationY`, and `sbe:rotationZ` data-type properties allow for specifying rotation information in degrees. Finally, the `sbe:scaleX`, `sbe:scaleY`, and `sbe:scaleZ` data-type properties allows for specifying size information in meters.
Table 5.2: The SBE ontology data-type properties.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building</td>
<td>buildingName</td>
<td>The building name to use in the UI.</td>
</tr>
<tr>
<td>Space</td>
<td>spaceName</td>
<td>The space name to use in the UI.</td>
</tr>
<tr>
<td></td>
<td>buildingUnit</td>
<td>The building unit where the space is located.</td>
</tr>
<tr>
<td></td>
<td>buildingFloor</td>
<td>The building floor where the space is located.</td>
</tr>
<tr>
<td>Transform</td>
<td>positionX</td>
<td>The x-position in meters with respect to an origin.</td>
</tr>
<tr>
<td></td>
<td>positionY</td>
<td>The y-position in meters with respect to an origin.</td>
</tr>
<tr>
<td></td>
<td>positionZ</td>
<td>The z-position in meters with respect to an origin.</td>
</tr>
<tr>
<td></td>
<td>rotationX</td>
<td>The x-rotation in degrees with respect to an origin.</td>
</tr>
<tr>
<td></td>
<td>rotationY</td>
<td>The y-rotation in degrees with respect to an origin.</td>
</tr>
<tr>
<td></td>
<td>rotationZ</td>
<td>The z-rotation in degrees with respect to an origin.</td>
</tr>
<tr>
<td></td>
<td>scaleX</td>
<td>The x-scale in meters with respect to an origin.</td>
</tr>
<tr>
<td></td>
<td>scaleY</td>
<td>The y-scale in meters with respect to an origin.</td>
</tr>
<tr>
<td></td>
<td>scaleZ</td>
<td>The z-scale in meters with respect to an origin.</td>
</tr>
</tbody>
</table>

Unlike GPS data that uses absolute values, quantitative indoor positioning information use relative values for position and orientation with respect to some origin. Different things (represented by ssn:Platform objects) may use different origins (represented by sbe:Origin objects) to specify their positioning information (represented by sbe:Transform objects). For an interaction proxy to support location-aware interaction with a given thing, it needs to register its coordinate system to the coordinate system of that thing. Supporting location-aware interaction is discussed in section 5.5.
5.1.2 Registration Service

SBEs are usually dynamic, where things can be added/removed and services can be created/destroyed over time. Automatic generation of SBE UIs requires maintaining up-to-date information about the incorporated things and the provided services. As the complexity of SBEs increases, it becomes a challenge to maintain such information manually. Referring to the automatic UI generation framework depicted in Figure 5.3, the registration service can help maintaining thing/service information automatically.

Thing/service information can be stored as an instance of the SBE ontology as mentioned in subsection 5.1.1. In order to maintain an up-to-date ontology, the registration service should keep track of all the available things/services and update the ontology accordingly. Different things/services may support different discovery service protocols (e.g. UPnP, SLP, CoAP, etc.), which utilize different standards and formats.

The registration service, as shown in Figure 5.5, consists of an API, a gateway, and a portal. The registration API provides a set of methods that allows for maintaining the information stored in the SBE ontology (e.g. register, unregister, etc). The registration gateway supports popular discovery service protocols and uses them to automate, or at least semi-automate, the thing/service registration process. The registration portal allows for registering things/services manually. This can serve as a last resort in case a thing or a service lacks support for any of the discovery service protocols supported by the gateway.

Although a discovery service protocol allows for collecting information about things/services automatically, it might not be able to provide all the required information (e.g. quantitative indoor positioning information for a thing). In such case, manual registration can help completing missing information about the registered things/services. Alternatively, a third-party agent may have the ability to collect such information automatically before storing it in the SBE ontology using the registration API.
5.2 Discovering an SBE

An SBE can provide many useful things/services to its users. Some of those things/services may act autonomously while others may require explicit user interaction. In order to interact explicitly with an SBE, a user should be aware of its existence in the first place. Otherwise, users may not be able to fully benefit from its incorporated things and provided services. A user should not be expected to have prior knowledge about an SBE before interacting with it. Instead, the interaction proxy should be able to discover the available SBEs on behalf of the user. Service discovery protocols such as UPnP and SLP are designed to support device discovery in local area networks. However, an SBE as a CPS should benefit from being connected to the Internet to support remote interaction. In order to achieve that, an SBE, as a first step, should support remote discovery.

Referring to the automatic UI generation framework depicted in Figure 5.3, the infrastructure should provide a discovery service that allows an interaction proxy to discover the available SBEs. The discovery service, as shown in Figure 5.6, consists of an SBE directory, a directory
agent, and an ontology agent. The SBE directory stores a record of basic information per SBE. This basic information (e.g. name, floor, building, GPS location, street, city, district, county, country, etc.) should facilitate searching for SBEs using different criteria. The information stored in the directory is obtained from the SBE ontology. It is the responsibility of the ontology agent to maintain up-to-date information in the directory by ensuring that the information record for each SBE remains consistent with the ontology. An interaction proxy relies on its discovery agent to discover the available SBEs by sending a discovery request to the discovery service. The directory agent is responsible for responding to such discovery requests with a list of SBEs that meet the criteria specified by the discovery agent in its request.

Figure 5.6: Discovering SBEs using a discovery service.

A users might be interested in SBEs in the surroundings or SBEs at a remote location. SBEs in user surroundings are often accessible through a Local Area Network (LAN). In such case, the infrastructure may provide a LAN-based discovery service, which can be used by the LAN-connected interaction proxies to discover the LAN-accessible SBEs. For an interaction proxy to discover SBEs at a remote location, the infrastructure needs to provide an Internet-based discovery service that maintains a global directory of SBEs. The discovery agent may benefit from context information to customize its discovery requests. For instance, a discovery agent may discover SBEs in user surroundings using an Internet-based rather than a LAN-based discovery service by sending a query that specifies the area of interest based on the current user location.
Once the discovery agent learns about the available SBEs, it can rank them based on context information. For example, the discovery agent may benefit from the availability of user location information to rank the obtained SBEs based on their distance from the user. Afterwards, the interaction proxy may select the SBE of interest autonomously given enough context information or present the user with a ranked list of SBEs for manual selection.

5.3 Learning about an SBE

Once the SBE of interest is selected from the set of discovered SBEs (as discussed in section 5.2), the next step is to learn about its capabilities. The interaction proxy needs to obtain a description of the incorporated things and the provided services before it can generate the SBE UI. This information should be available in a machine-readable format that can be interpreted by any interaction proxy regardless of its capabilities or the technology it uses. Learning about an SBE can take place through the learning service, which is illustrated in subsection 5.3.1. An approach to describe things/services for UI generation purposes is presented in subsection 5.3.2.

5.3.1 Learning Service

Referring to the automatic UI generation framework depicted in Figure 5.3, the infrastructure should provide a learning service that allows an interaction proxy to learn about the SBE of interest. The learning service, as shown in Figure 5.7, consists of a thing/service descriptions repository, a description generator, and a description agent. The thing/service descriptions repository stores a record of information per thing/service. This information should include the associated SBE(s) besides a complete description of the thing/service attributes and functionalities. The information stored in the repository is obtained from the SBE ontology. It is the responsibility of the description generator to maintain up-to-date information in the repository by ensuring that the information record for each thing/service remains consistent.
with the ontology. An interaction proxy relies on its learning agent to learn about the SBE of interest by sending a request to the learning service. The description agent is responsible for responding to such requests with a list of relevant things/services along with their complete descriptions.

![Diagram](image)

**Figure 5.7:** Learning about an SBE using a learning service.

### 5.3.2 Thing/Service Description

The description generator, as mentioned in subsection 5.3.1, is responsible for generating a machine-readable description of the attributes and functionalities of each thing/service. This description is intended to be used for UI generation purposes. As shown in Figure 5.7, the description generator relies on the SBE ontology to infer the required information before generating the descriptions and storing them in the thing/service descriptions repository for later use. This subsection presents a proposed approach to describe things and services. The main goal behind this proposed approach is to allow for providing a unified thing/service description that can be turned into a UI of the designated type by any interaction proxy, regardless of the UI type(s) it supports. Therefore, the description must be generic enough to avoid placing constrains on the UI type(s) that can be generated while being informative enough to allow for generating usable UIs.

From an object-oriented perspective, things/services are objects with state and behavior. The state of a thing/service can be determined by the values of its attributes while the
behavior can be defined by the actions it can take. In order to support explicit user interaction with those objects (i.e. things/services), an SBE UI should allow for monitoring their attributes and triggering their supported actions.

Thing/Service Descriptors (TSD) is a proposed approach to describe things and services in an SBE in a machine readable format. It adopts the concept of a channel to represent the communication streams between things/services and the UI. A thing/service is represented as a set of data channels and action channels. A data channel is associated with a thing/service attribute and represents a data stream while an action channel represents an action stream. A UI should be able to reflect the state of a thing/service using the values obtained from the appropriate data channels and should as well enable the users to trigger actions by sending the appropriate commands over the appropriate action channels. The event Channel is another type of channels that a thing/service may use to inform the user about an event (e.g. device overheating) or an error (e.g. sensor failure). The UI should monitor the event channels and notify the user whenever an event/error message is received.

Providing a well-formed and informative description of each thing/service in an SBE should allow the UI generator to learn about that SBE. The proposed TSD relies on the following elements to describe an object, whether that object is a thing or a service:

- **id**: uniquely identifies the object;
- **type**: specified whether the object is a thing or a service;
- **name**: a convenient name to use in the UI to refer to the object;
- **dataChannels**: a list of specifications of the object’s data channels;
- **eventChannels**: a list of specifications of the object’s event channels;
- **actionChannels**: a list of specifications of the object’s action channels;
Things have physical embodiments and their positioning information can be crucial to the interaction experience. Therefore, the proposed TSD relies on the following additional elements to describe the positioning information of a thing:

- **origin**: a unique id that specifies a coordinate system;
- **position**: thing position under the specified coordinate system;
- **rotation**: thing rotation under the specified coordinate system;
- **Scale**: thing size (footprint) under the specified coordinate system.

The proposed TSD uses the following elements to describe a channel:

- **id**: uniquely identifies the channel;
- **format**: specifies the data format using Internet media types;
- **name**: a convenient name to use in the UI to refer to the channel;
- **endpoint**: specifies the protocol, host, port, path, authentication, etc.;

The elements mentioned above can describe a channel, whether this channel is for data, events, or actions. Both data and event channels receive their messages from things/services. Conversely, action channels receive their messages from the UI. Therefore, it is the responsibility of the UI to ensure that those messages are valid by applying the appropriate constrains. The proposed TSD relies on the following additional elements to describe an action channel:

- **type**: specifies the command type (e.g. boolean, integer, etc.)
- **minlength** specifies the minimum length for a string-type command;
- **maxlength** specifies the maximum length for a string-type command;
• \textit{minValue}: species the minimum value for a numeric-type command;

• \textit{maxValue}: species the maximum value for a numeric-type command;

• \textit{enumValues} specifies the possible values for an enumeration-type command;

The proposed TSD provides the required element to describe the various aspects of a thing/service for UI generation purposes. It allows the UI generator to learn about the attributes of a thing/service as well as the actions it can take. Moreover, it provides information about the constraints associated with those actions. Yet, it is generic enough and does not assume or require the use of specific UI types. The learning service \textit{\text{(subsection 5.3.1)}} relies on its description generator component to generate a TSD for each thing/service. The information required to generate the TSD is obtained from the \textit{SBE ontology}. The learning service can identify missing information in the ontology during the description generation process before providing a report with suggestions to fix the ontology. The generated TSDs are stored in the \textit{thing/service descriptions} repository. The description agent should be able to provide those descriptions in various formats using Internet media types (e.g. \texttt{application/json;charset=UTF-8}) to facilitate interpretation by any interaction proxy.

\section{Generating an SBE UI}

Once the interaction proxy obtains the TSD for each thing/service in the SBE of interest (as discussed in \textit{section 5.3}), the next step is to generate the SBE UI. This involves interpreting those descriptions and mapping them to a UI of the designated type based on the available I/O capabilities. Referring to the automatic UI generation framework shown in \textit{Figure 5.3}, the \textit{learning agent} is responsible for obtaining the TSDs from the learning service before feeding it to the \textit{UI generator}, which in turn generates the SBE UI.

Different interaction proxy types have varying I/O capabilities. Consequently, the UI types that they can support will vary as well. The UI generator needs to decide on which UI type
to generate based on the available I/O capabilities before generating a UI that is compatible with the interaction proxy platform. This means that the implementation of the UI generator will vary from one interaction proxy type to another. In fact, the UI generator can be split into two components (a mapper and an implementer) as shown in Figure 5.8. The mapper maps the TSDs to a UI description of the designated UI type before the implementer converts that UI description into a fully-functioning UI.

Figure 5.8: Generating a UI from thing/service descriptors.

Splitting the UI generator into a mapper component and an implementer component allows for reusability, where the same mapper can be used by different interaction proxy types given that they are all willing to provide the same UI type. For instance, a mapper that maps a set of TSDs into a GUI description can be used by any interaction proxy that needs to provide a GUI. Only the implementer component will need to be tailored for each interaction proxy platform (e.g. Android, Windows, etc.).

5.5 Adapting an SBE UI

Context awareness can be crucial to user interaction in SBEs. As the number of things and services in an SBE increases, the corresponding UI can get more complex and thereby less usable. Context information can help address this problem by allowing an interaction proxy
to adapt the UI to the current context, which can facilitate user tasks and enhance their experience of the SBE. An interaction proxy may obtain context information from several sources either directly using its own capabilities or indirectly through the capabilities of the SBE. Context information can help adapting the UI to prioritize certain things or services over others based on the current user task, which can be detected using sensors or predicted using predefined rules.

An SBE is a physical environment and the things it incorporates have physical embodiments. Therefore, location information can be of great significance to a context-aware SBE UI. Consider a building with multiple SBEs that incorporate numerous things. Instead of requiring the user to navigate the UI manually, a context-aware UI can automatically adapt to the current user location. Recognizing the SBE where the user is currently present can be achieved through tag devices that are embedded in that SBE. An interaction proxy may query the ontology about the available tag devices and their corresponding SBEs before searching for those tags in the surrounding SBE. Once the interaction proxy finds one or more tag devices, it can use them to identify its current location.

As a CPS, an SBE can benefit from being connected to the Internet to access remote information that can help better understanding the current context (e.g. online calendars, social networks). Moreover, it can use cloud services to perform resource intensive tasks (e.g. data storage and visualization, machine learning, etc.). This provides several opportunities that were not applicable/feasible in closed systems.

5.6 Maintaining UI/SBE Consistency

Once an interaction proxy provides the UI for an SBE, users may use it to interact with that SBE. An SBE may have several UIs provided by different interaction proxies. Those UIs should be consistent and reflecting the current state of the SBE. This way a user may start a given task using one UI and resume that task using another. The state of an SBE
may change as a response to an autonomous action or an explicit user interaction. In either case, this change should be reflected in all UIs of that SBE.

Two widely used communication models are request-response and publish-subscribe. In WoT, as discussed in subsection 2.3.3, things utilize well-established Internet protocols and expose their functionalities in the form of RESTful services, which allows for easy integration of heterogeneous things. However, maintaining consistency between things and their UIs under a request-response communication model can be challenging. For a UI to stay consistent with an SBE, it needs perform periodic pull requests. Consequently, that UI may not reflect the current state of the SBE until the next pull request. If the frequency of pull requests is $t$ then the UI may stay inconsistent with the current state of the corresponding SBE for up to $1/t$. Increasing the frequency of pull requests may help minimizing inconsistency periods. However, it can introduce a significant communication overhead. Therefore, relying on pull requests to maintain consistency between an SBE and its UIs may not be feasible, especially as the number of UIs and the things/services increases. The publish/subscribe communication pattern provide the opportunity for better communication scalability compared to pull requests. UIs can subscribe to the streams of interest and receive instant updates. This can help maintaining consistency between an SBE and its UIs while reducing the network overhead.
Part III

Evaluation and Discussion
Chapter 6

Simulating MQTT-Based Communication

As discussed in section 5.6, network glitches can degrade the responsiveness of the SBE and impact the consistency between things and their UIs, which can in turn degrade the user experience. Therefore, it is crucial to ensure reliable and responsive communication between thing, services, and UIs of an SBE. Things are usually constrained devices with limited computation and communication resources. Moreover, they might be battery-powered with a limited lifetime before they need recharging. Therefore, research on minimizing the overhead and reducing power consumption in constrained devices has received a significant attention. Communicating redundant or irrelevant data is infeasible at the device level as well as the network level, especially as the number of interconnected devices increases. Therefore, some researchers have proposed a variety of techniques, such as data prediction \cite{Jain2004, Santini2006, LeBorgne2007, Zhao2012} and adaptive sampling \cite{Alippi2010, Szczytowski2010, Gupta2011}, to reduce the amount of communicated data.

Several communication protocols have been proposed and adopted to support data exchange between interconnected devices. However, with the introduction of constrained devices came
the need for lightweight communication protocols. The Message Queue Telemetry Transport (MQTT) Protocol is a lightweight communication protocol that is widely adopted by different parties to support communication in the IoT. It runs over TCP and acts as a pipe for binary data. Therefore, it supports reliable connection-oriented communication and provides flexibility in communication patterns. MQTT is a publish-subscribe-based messaging protocol, where each message has a topic and a payload. As depicted in Figure 6.1, a message broker receives messages from publishers and distributes them to the subscribers based on the topic of the message.

![Figure 6.1: The publish/subscribe communication pattern of the MQTT protocol.](image)

Designing an SBE is a complex process with many decisions to be taken to ensure a successful system. Prototyping is one way to evaluate different aspects of an SBE before the actual implementation. However, besides the cost involved, it is usually tricky to evaluate a system based on prototypes. Fortunately, simulating SBEs can offer a cheaper solution compared to building prototypes. Moreover, it offers the flexibility of changing a variety of parameters and evaluating different candidate designs. Simulating an SBE requires simulating several entities including the devices, the communication network, and the communication pattern as well as the context and the behavior of the users.

Maintaining consistency between an SBE and its UIs requires providing a reliable and responsive communication network. A simulation tool was developed to simulate MQTT-based communication networks [Handosa et al., 2017a]. This tool can facilitate the design of SBEs, or IoT systems in general. Simulating MQTT-based communication requires simulating the
communicating nodes as well as the network that connects them. The simulation tool uses the NS-3 network simulator to simulate different types of networks with different parameters. Each node is emulated by a lightweight virtual machine, namely a Linux container. A broker node hosts an implementation of the MQTT broker, namely Mosquitto [Light, 2013], while a client node hosts an application that can publish messages and/or subscribe to topics. The virtual machines are interconnected through the simulated network using a set of virtual bridges and tap devices as shown in Figure 6.2.

![Figure 6.2: Virtual machines connected through a simulated network.](image)

The MQTT simulation tool can automatically create the nodes along with the simulated network that connects them according to a set of parameters that can be adjusted such as the number of publishers, the number of subscribers, the number of topics, the frequency of messages, the required quality of service (QoS), etc. For instance, Figure 6.3 shows the average delay using a simulated carrier-sense multiple access (CSMA) network with different numbers of nodes and different QoS levels.
Figure 6.3: Results obtained from the MQTT simulation tool.
Chapter 7

Supporting User Interaction

The interaction between users and things in an SBE can take several forms. No single form of interaction is perfect for every scenario. Although GUI and voice-based UIs are commonly used nowadays to support user interaction with things in SBEs, they suffer from a number of limitations as discussed in section 3.4. In this chapter, we explore alternative UI types and interaction techniques to support user interaction with SBEs. Meanwhile, we explain how to benefit from the proposed framework for explicit user interaction (Figure 5.1) to better understand the possible interaction alternatives and identify the promising ones.

The proposed UIs and interaction techniques presented in this chapter were explored using the FutureHAUS testbed. The FutureHAUS is a multidisciplinary research project with a focus on building prefabricated modular structures that integrate smart technologies. It allows for creating and connecting various things with embedded sensors and actuators, collect data, explore patterns of use and test different alternatives to support user interaction. The FutureHAUS has participated in the Solar Decathlon Middle East (SDME) 2018 and won the first place. We have received positive feedback from the competition juries and the event visitors about the incorporated capabilities, the implemented features, and the provided support for user interaction through multimodal UIs and innovative interaction techniques.
The FutureHAUS incorporates numerous sensors that allow for monitoring the state of the house and its inhabitants as well as actuators that can change the state of the house. For instance, the bathroom module has a load cell sensor mounted under the floor tile to detect user presence while preserving user privacy compared to the use of a camera. Presence information can be used to trigger autonomous actions such as turning the lights on/off or playing music. Furthermore, the sensor reading can be used to identify a user based on the measured weight before triggering personalized actions.

The flexibility that the FutureHAUS offers by design allows for realizing the concept of “aging in place”. The incorporated actuator can adjust various aspects of the house to meet user needs. For instance, a sink or a kitchen cabinet can be moved up and down. This makes it easily reachable and more convenient for children and/or people with disabilities, where they no longer need to ask for help as the house itself can help. The bathroom module has a proximity sensor that can measure the height of the user. As an example of implicit interaction, this information can be used to adjust the height of the sink autonomously rather than requiring the user to do that explicitly using a physical switch or other UI.

Regarding the support for explicit user interaction, the house can be controlled using smartphones and tablets as well as wall-mounted touchscreens that are incorporated in each room. In addition, voice-based interaction is supported using smart speakers (e.g. Amazon Echo). Besides the popular GUIs and voice-based UIs, we have explored other means of interaction such as gesture-based and MR-based UIs.

The following sections provide some examples of supporting user interactions with the FutureHAUS as an SBE. In section 7.1, we present a proposed gesture-based interaction technique to control lights in indoor settings. Another example of using gesture-based UIs is presented in section 7.2. We explore the use of MR-based UIs in section 7.3 and present a proposed approach to extend embodied interaction in MR environments in section 7.4.
7.1 Gesture-Based Light Control

Lighting plays an essential role in supporting user tasks as well as creating an ambiance in the illuminated space. Therefore, controlling lights can be regarded as one of the basic tasks that users will often need to perform as part of their interaction with SBEs. Traditional lighting systems allow users to control light bulbs using switches; flipping the switch turns the light on/off. However, with recent advances in lighting technologies, a lighting control system is no longer as simple as it once was in the past. The Light Emitting Diode (LED) technology provide new opportunities as well as challenges. LED lights allow users to adjust different light parameters including color and intensity thus providing a wider range of possible lighting configurations.

The complexity of light control arises mainly from two factors. First, the increasing number of light sources, where a LED-based lighting system can consist of tens or even hundreds of individually controllable light sources [Aliakseyeu et al., 2012]. Therefore, installing a switch per light source, as shown in Figure 7.1, is no longer a feasible solution as it does not scale well. Second, the adjustable lighting parameters (e.g. color and intensity) cannot be configured using traditional switches, which implies the need for a new lighting control interface.

![Figure 7.1: A real-world example of a fairly complex set of light switches and regulators.](image-url)
Besides saving energy [Martirano, 2011], a lighting system should achieve two main goals. First, support a variety of lighting features and functionalities. Second, provide a convenient light control UI. Current lighting systems have mainly focused on achieving the first goal and relied on well-established interaction technologies (e.g. GUI) to achieve the second goal. Some lighting system vendors provide smartphone apps to control lights. However, mapping light sources positioned in a 3D space to a 2D GUI can confuse the user, especially as the number of light sources increases and their distribution in space becomes more complex.

Philips provides a smartphone app (Figure 7.2a) as well as a physical switch with four programmable physical buttons (Figure 7.2b) to support user interaction with Philips lighting systems. Following our proposed approach trying to understand how the smartphone app and the physical switch fit in the interaction context, reveals that Philips support user interaction with lights through two alternative interaction proxies. The smartphone is a user-side exposed interaction proxy while the physical switch is an SBE-side exposed interaction proxy. Both rely on touch/press to provide input and rely on the lights themselves to provide output. In addition, the smartphone can provide the output through a GUI, allowing for remote interaction. Both solutions suffer from drawbacks, where both are exposed interaction proxies acting as explicit barriers between users and lights. While the smartphone app can support advanced functionalities, using it to perform simple lighting control tasks (e.g. turning lights on/off) may not be always feasible. Compared to a smartphone app, a physical switch allows for more efficient interaction but provide a very limited UI.

Given the drawbacks associated with the currently adopted interaction techniques to support lighting control tasks, it is crucial to adapt those techniques to address their drawbacks or explore other alternatives that have the potential to provide an enhanced interaction experience. Achieving a seamless transparent interaction between users and lights may take place through an SBE-side hidden interaction proxy. Referring to Table 4.1, two candidates are voice-based and gesture-based UIs. A voice-based UI allows for intuitive and efficient interaction, especially for simple lighting control tasks (e.g. ”turn the kitchen light off”). However, it may not be a wise choice for controlling more complex lighting systems with
various parameters (e.g. a room with tens of color-adjustable light sources) as it can be challenging for the user to memorize the mapping between the voice commands and the corresponding light sources not to mention the voice commands to adjust the color and intensity. On the other hand, the gesture-based control provides a promising opportunity to leverage other capabilities of the human body, allowing the user to simply point to the light source rather than recall its name. In [Petersen and Stricker, 2009], the authors described a user study showing that 80% of the participants preferred to use gestures over more traditional methods such as GUIs. The participants had no problem completing their tasks after the first or second try.

In [Mrazovac et al., 2011], the authors used a sensing glove to provide 3D light control. Their system allows for switching lights on/off as well as dimming them. Using a radio frequency transmitter, the glove sends accelerometer data to a remote module that translates it into lighting commands. The system uses the sensing glove to control a specific light source and there is no support for selecting different light sources. Moreover, only the intensity of the light source can be controlled but not its color. Furthermore, the glove is a user-side
exposed interaction proxy that acts as an explicit barrier between the user and the lights not to mention the practical complications associated with wearing a wired glove in everyday life to control lights.

The following subsections describe our proposed gesture-based approach to control color-adjustable LED-based lights [Handosa et al., 2017b]. Using the FutureHAUS as a testbed, we developed a system to control lights in the FutureHAUS living room. The developed lights control system uses a tracking device to capture user gestures. Compared to the use of a glove as in [Mrazovac et al., 2011], the tracking device is an SBE-side hidden interaction proxy that allows for hands-free seamless interaction between users and lights. The proposed approach allows for selecting individual light sources before turning them on/off and/or controlling their color/intensity.

### 7.1.1 Approach

Light control tasks can be categorized as either basic tasks or advanced tasks. Basic light control tasks include selecting one or more light sources before turning them on/off or adjusting their color and/or brightness if applicable. More advanced tasks might include defining lighting preferences, defining static or dynamic lighting patterns, programming specific times for different lighting patterns, or defining rules for initiating a specific lighting configuration when certain conditions are met. A taxonomy of the basic light control tasks is shown in Figure 7.3. Turning a light source off can be achieved by setting its brightness to zero while turning it on can be achieved by setting its brightness to a non-zero value.

![Figure 7.3: A taxonomy of basic light control tasks.](image-url)
Basic light control tasks are often performed frequently where the advanced ones are usually performed rarely. Our proposed light control approach focuses on the basic tasks allowing a user to perform any of them using simple hand gestures. Compared to a GUI, our approach can reduce the number of required interactions. This provides a promising alternative to perform simple light control tasks, where it might be inefficient to use a GUI or a voice-based UI. The approach defines a set of hand gestures to represent different light control commands and relies on the light sources themselves to provide feedback to user commands. The gestures are captured using an SBE-side interaction proxy (e.g. a camera-based tracking device), which can capture the gestures of multiple users. Consequently, the approach needs to support conflict resolution to handle situations where multiple users try to interact with the light control UI simultaneously. In order to achieve that, the approach is designed to respond to a single user at a time. Once a user claims control, the UI will respond to that user only until that user releases control. Afterward, another user may claim control and start using the UI. In order to control lights, the user will use both hands, one as a selection hand and the other as a manipulation hand. The following are the set of user actions that the gesture-based light control UI can recognize:

- **Claiming control**: the user can claim control by raising either hand above the head level and making a closed hand gesture. Once the control is claimed that hand becomes the selection hand and the other hand becomes the manipulation hand. No other user can use the UI until that user releases the control.

- **Releasing control**: the user can release control by making an open hand gesture using the selection hand. The control is released automatically when the user body is no longer tracked by the UI (e.g. the user left the room). Once the control is released, any selected light sources are deselected.

- **Selection**: the user can select one or more light sources by simply pointing at them using the selection hand. The selection ray cast, as shown in Figure 7.4, starts from the position of the user’s head and goes through the selection hand. To simplify the
user’s task the UI will select a light source as long as it is within a predefined distance of the selection ray cast. Selected light sources do not have to be contiguous. They can be scattered or grouped in separate regions. The user can deselect lights by releasing control.

- **Switching hands**: a user can switch selection and manipulation hands without losing the current selection. In order to achieve that, the user raises the manipulation hand above the head and makes a closed hand gesture with it before opening the selection hand. At that point, the old manipulation hand becomes the new selection hand and the old selection hand becomes the new manipulation hand. The new selection hand can now be used to add more light sources to the current selection. In this way, the user can make use of both hands to select multiple light sources in various directions. This should be more convenient to the user compared to requiring the user to use the same hand to select all lights sources, which might be tricky to the user and requires body twists or turning around if the light sources are located in different directions around the user.

- **Manipulation**: the user can use the manipulation hand to manipulate the currently selected light sources. The UI assumes three virtual sliders corresponding to the three color components, hue, saturation, and brightness. The user can change the value of a slider by closing the manipulation hand and moving it along the slider before opening the coloring hand at the desired value. The user closes, moves, and opens the manipulation hand as if she is catching, dragging, and releasing the virtual slider and sees instant feedback as the selected lights change their color in response to the change in slider value. The user controls the hue slider by moving the manipulation hand left and right, controls the saturation slider by moving the manipulation hand forward and backward, and controls the brightness slider by moving the manipulation hand up and down. Requiring the user to close and then open the manipulation hand to change the value of any of the three color components helps to avoid accidental coloring due to
unintended hand movements. Once the user closes the manipulation hand the initial movement direction of that hand specifies which of the three sliders is activated. The user can control only one slider at a time and must open the manipulation hand to release that slider before activating another one. This simplifies the user’s tasks by eliminating the possibility of unintended value changes of a color component while the user is changing another.

Figure 7.4: Selecting a light source.

7.1.2 Implementation

The developed light control system provides two alternative methods for controlling lights, a 2D GUI and a 3D gesture-based UI. The system as shown in Figure 7.5 consists of three clients and a broker that facilitates communication between them. The 2D UI client runs on a computing device with a 2D display (e.g. smartphone) and provides a 2D graphical
representation of different light sources together with a color palette. The user can use that palette to select a color before applying it to one or more light sources. The 3D UI client is responsible for interpreting user’s gestures into lighting control commands before sending them to the Digital Multiplexer (DMX) client. The DMX client is responsible for translating the received commands into DMX commands before sending them to the DMX controller.

![Diagram of the lighting control system]

The system uses a Microsoft Kinect device as a tracking device to capture user gestures. The DMX controller device can control a set of light sources individually. Communication with the DMX controller takes place using the DMX512 protocol. The MQTT broker facilitates communication between the 2D UI client, the 3D UI client, and the DMX client. Communication between clients takes place using the MQTT protocol.

The system was deployed and tested in the FutureHAUS living room, as shown in Figure 7.6, with 36 individually controllable LED segments mounted in the ceiling forming a rectangle with 10 segments on each of two opposite sides and 8 segments of each of the other two opposite sides. The locations of those light segments in 3D space were fed to the system upon deployment. The selection tolerance can be customized to adjust selection sensitivity. The basic rule is that the nearest light source to the selection beam is selected if its distance from the selection ray cast is shorter than the specified tolerance value. The granularity of the hue, saturation, and brightness sliders can be customized as well. Increasing the slider granularity allows for more accurate control of its value. However, this will increase the length of the slider, which may require the user to do longer/multiple drags to set the desired value.
7.1.3 Evaluation

The developed light control system depends on the lights themselves to provide feedback to the user. Thus, the lights are expected to reflect user actions instantly. A noticeable delay in response to user commands can degrade the user experience. For instance, receiving late feedback for a previous set hue command while performing a set brightness command can confuse the user. Therefore, the responsiveness of the system is critical for the system’s usability.

The responsiveness of the system can be evaluated by estimating the delay (latency) between the time at which the user makes a given gesture and the time at which the user receives the corresponding feedback. In order to estimate the delay, a camera was used for video recording of various user actions. Exploring the frames of the captured videos revealed that it takes at most five frames to receive feedback after the user action takes place. Figure 7.7 shows six consecutive frames from one of the captured videos. In Figure 7.7a, the user points to a light source with an open hand, which should not trigger any action. In Figure 7.7b, the user closes his hand, which results in claiming control of the system, defining the user
right hand as a selection hand, and selecting the light source that the user is pointing at. In Figure 7.7c, d, and e, the user is waiting for feedback. In Figure 7.7f, the user receives visual feedback in the form of a change in the color of the selected light source.

Figure 7.7: Video recording frames for selecting a light source.

Assuming that frame (a) was captured at time 0 and frame (b) was captured at time $T$, then the user gesture takes place at time $0 < t \leq T$. Similarly, if the frame (e) was captured at time $4T$ and frame (f) was captured at time $5T$, then the feedback has occurred at time $4T < t \leq 5T$. Consequently, the delay $d$ is $3T < d < 5T$. The captured video recordings have a frame rate of 29.97 frames per second and $T = 1/29.97$. Consequently, the total system delay ranges between approximately 100 and 167 milliseconds.

The system delay consists of a sequence of processing and communication steps that take place between user’s action and the corresponding feedback. The total delay $d \geq d_1 + d_2 +$
\(d_3 + d_4 + d_5\), where \(d_1\) is the time it takes the Microsoft Kinect device to capture the scene, \(d_2\) is the time it takes the 3D UI client to process the Kinect data and infer the user’s command, \(d_3\) is the network communication delay between the 3D UI client and the DMX client, \(d_4\) is the time it takes the DMX client to translate the received command into a DMX command, and \(d_5\) is the time it takes the DMX controller to configure the lights accordingly to the received DMX command. The delays \(d_1\) and \(d_5\) are device specific and beyond our control. The measured average values of \(d_2\) and \(d_4\) are approximately 0.83 and 2.57 milliseconds, respectively.

The MQTT protocol supports three Quality of Service (QoS) levels. The message delivery for QoS-0, QoS-1, and QoS-2 are at-most-once, at-least-once, and exactly-once, respectively. Test results have shown that the average delay for the QoS-0, QoS-1, and QoS-2 are approximately 6.41, 23.18, and 66.21 milliseconds, respectively. Although QoS-0 provides the smallest average delay, it can result in losing messages and hence dropping user commands, which is unacceptable for this system. QoS-2 guarantees exactly-once message delivery however its associated delay is significantly large compared to the other two. Although the use of QoS-1 can result in duplicate messages, it was selected because it guarantees message delivery while its average delay is much better than that of QoS-2. Duplicate messages, if any, should not affect the validity of the system. However, they will result in unnecessary repeated processing at both the DMX client and the DMX controller, which can be avoided by filtering them out upon arrival.

The developed system is extendable and allows multiple users to control lights using either the 3D UI (i.e. the gesture-based UI) or through the 2D GUI. The proposed gesture-based light control UI allows the users to perform the frequent light control tasks using simple hand gestures. Meanwhile, the developed system is flexible enough to incorporate the GUI to support remote interaction as well and to enable users to perform advanced light control tasks that cannot be accomplished using simple hand gestures.
7.2 Gesture-based Wall Movement

The FutureHAUS testbed allows for moving walls to expand one room at the expense of another. This is achieved using actuators that can be controlled through a UI. Traditional candidate UIs may include a wall-mounted physical switch, a wall-mounted touchscreen, or a smartphone app. All of these alternatives rely on exposed interaction proxies that act as explicit barriers between the user and the wall. The physical switch and the wall-mounted touchscreen are both SBE-side interaction proxies that are preconfigured and ready for instant use while the smartphone is a user-side one that may require more interactions to configure it before it can be used to control the wall.

Thinking of a more intuitive interaction to move a wall may lead to the use of an interaction proxy that is SBE-side (i.e. no configuration before interaction is required at the user side) and hidden (i.e. allows for a transparent and seamless interaction). Referring to Table 4.1, two candidates are voice-based and gesture-based UIs. For safety reasons, a voice-based UI might not be a wise choice as false positives may lead to accidental wall movement that can harm people or cause physical damage.

A gesture-based UI can turn the SBE into an exoskeleton that allows a user to move the walls (or other heavy objects) by applying limited or no force. Rather than reaching a physical switch, a user may push/pull the wall from any point turning the whole wall into an interactive surface. Meanwhile, a tracking device may capture user actions and interpret them as commands to move the wall. A typical implementation may require the user to touch the wall for a couple of seconds to activate the movement before the wall follows the user’s hand. The user may stop the wall by removing the hand from its surface. The same tracking device may monitor the space to detect obstacles or other humans and deactivate wall movement automatically to ensure the safety of the SBE occupants.
7.3 MR-based Virtual Assistant

In non-domestic built environments, users may need assistance to locate and access resources. Public places like museums and libraries can benefit from virtual assistants to provide guidance to their visitors. Referring to Table 4.2, an interaction proxy may provide a virtual assistant in the form of a graphical, audio, or MR-based UI. A GUI or an audio-based UI hosted on a smartphone may provide valuable information about navigating and locating resources in a public place. However, mapping this information to the 3D space can be tricky to the user.

With recent advances in MR technologies, there is a great opportunity to use MR headsets to support user interaction in SBEs. MR is a rapidly increasing field of study because it has the unique advantage of enabling natural movement and interaction with both the physical and the virtual world in three dimensions, thereby using affordances for a better sense of presence. It blends virtual objects into the physical space allowing for an unprecedented interaction experience. Incorporating holograms in MR enables one to visualize and work with the digital content as part of the real world. Holograms are responsive to the user and the surrounding environment, providing a rich interaction experience.

MR can be used to bridge real and virtual library [Gračanin et al., 2017]. A virtual avatar can be projected into the physical space to play the role of a librarian. Using SBE-side devices to track the user and a digital system that is aware of the resources and their availability, the virtual avatar can interact with users and guide them to the resources of interest. Such system demonstrates the user of a multiple-device interaction proxy that leverages both tracking devices (i.e. SBE-side devices) and user-side devices (i.e. MR headset). Compared to the use of a GUI or an audio-based UI, using virtual objects that blend into the physical environment allows for a more intuitive interaction as it enables the user to develop a clear mental model that maps the UI to the physical surroundings.
7.4 Extending Embodied Interaction

In the literature, some researchers proposed the use of Virtual Reality (VR) to provide a 3D UI for SBEs. Compared to a 2D GUI, a VR-based UI can provide a more intuitive mapping between the virtual UI elements and the corresponding physical things located in the 3D space. In [Borodulkin et al., 2002], the authors created a 3D virtual environment for home automation control to replace abstract objects representation and hierarchical windows used in the Window Icon Menu Pointer (WIMP) interfaces and provide a realistic view of the house with its structured components. In [Han et al., 2010], the authors proposed an Internet-based home automation system that allows users to control smart objects through a 3D virtual model of the physical space.

Although a 3D virtual model can provide a more realistic interface for an SBE compared to the flat 2D GUIs, it still acts as an explicit barrier between the users and the SBE. A VR-based UI will immerse the user in a virtual environment that blocks the user’s perception of the physical surroundings, especially if a virtual reality headset is used. This makes it tricky for the user to navigate the virtual environment with actual body movement.

Alternative navigation techniques allow a user to navigate the virtual space without actually walking around in the physical space [Kim et al., 2015]. However, such techniques do not provide the same level of seamless interaction and engagement with the virtual environment as the intuitive physical body navigation can do. Moreover, some users may experience motion sickness due to the variation between their physical motion and the corresponding visual perception [Hettinger and Riccio, 1992].

The recent advances in MR technologies helped the MR headsets to gain popularity and to become in reach to the users. An MR-based UI can augment things in an SBE with UI elements in the form of holograms, as shown in Figure 7.8. This allows for a more intuitive UI navigation compared to a GUI, where a user can navigate to the UI for a given thing by simply looking at that thing. Compared to a VR-based UI, an MR-based UI does not block
the user from perceiving the physical surroundings. Therefore, a user can still navigate the physical space by simply walking around while wearing the MR headset.

Different MR headsets support user interaction in a variety of ways, where a user may provide input using physical controls, voice commands, and/or hand gestures. For instance, the ODG smart glasses device supports user input through on-device buttons/trackpad, a Wireless Finger Controller (WFC) with motion/gesture functionality, and a Wireless Bluetooth Keyboard with multifunction command keys. Another example is the Microsoft HoloLens device, which uses spatial mapping to place holograms in the surrounding physical space and supports user input through voice commands, a single-button Bluetooth device (HoloLens Clicker), and a limited set of hand gestures (bloom and air-tap).

To achieve full immersion in an MR environment, the support for user interaction should be as natural and intuitive as possible. The spatial awareness of an MR headset (e.g. Microsoft HoloLens), allows for a great degree of freedom regarding recognition, movement, and exploration of the MR environment. However, the limited input capabilities of an MR headset prevents it from supporting intuitive interaction, where a user can interact with holograms naturally as if they were physical objects.
Expanding on the Microsoft HoloLens device as one of the popular examples of MR headsets, we find that its interaction concept is based on voice commands, gaze tracking, hand tracking, and gesture recognition. The Microsoft HoloLens device provides a limited interaction experience, which can degrade the user’s perception of the MR environment. This is due to the limitations induced by its gaze tracking, hand tracking, and gesture recognition as explained below.

- **Gaze tracking.** The Microsoft HoloLens device uses its orientation as an indicator of its user’s gazing direction. This assumption is not always true as a user may gaze at different directions while maintaining the same head orientation. Therefore, a cursor hologram is usually used to help the user perceive the gazing direction assumed by the HoloLens device. Adding this extra hologram to the MR scene may not be the best way to support natural and intuitive interaction.

- **Hand tracking.** The HoloLens device can track the position of the user’s hand if (1) the user’s hand is within its tracking space, which is limited, and (2) the user’s hand is in the ready state, which is a closed fist with the index finger pointing up as shown in Figure 7.9a. Maintaining the hand in the ready state and within the limited tracking space of the HoloLens device can be tricky and inconvenient to the user, especially for long interaction scenarios. Moreover, HoloLens cannot discriminate between left and right hands. In fact, HoloLens tracks a hand as a disjoint object floating in space with no information about its side nor whether it belongs to the user or not. Consequently, a HoloLens device may track the hand of a person other than the user (if that hand is within the tracking space of the HoloLens device) and trigger actions accordingly, which can cause interaction conflicts in a collaborative environment with multiple users wearing HoloLens devices and working in proximity.

- **Gesture recognition.** HoloLens can recognize two core gestures only (bloom and air-tap). The bloom gesture is reserved by the system to act as a “Home” gesture that allows a user to go back to the “Start Menu”. Therefore, air-tap is the only gesture that
can be programmed to trigger actions in a HoloLens app. The air-tap is a transition between two recognizable hand states (ready and press) as shown in Figure 7.9. Performing an air-tap has the same effect as clicking the button of the HoloLens Clicker. Considering that a user will need to perform the air-tap while her hand is within the tracked space of the HoloLens to trigger a click action, it might be easier and more efficient to trigger that action using the HoloLens Clicker.

As discussed above, the Microsoft HoloLens headset provides a limited interaction experience, where (1) both the hand tracking and the gesture recognition require the user’s hand to be within the limited space that the HoloLens device can track and (2) the gaze tracking follows the orientation of the HoloLens device rather than the actual gaze direction of the user, requiring the user to adjust the head orientation towards the object of interest rather than simply gazing at it. These preconditions add a limitation to the possible space of interaction and create the need for not necessarily natural behavior patterns in order to interact with objects in a given environment. The lack of custom gesture recognition and full-body tracking (or at least discriminating left and right hands) limits the possible range of interaction. Natural interaction patterns such as using both hands at the same time to interact with multiple holograms simultaneously are not (or only to a certain degree) possible.
7.4.1 Approach

Observing the HoloLens user using tracking devices, such as the Microsoft Kinect device, allows for full-body tracking and identification of different body parts (not just hands). This information can help in developing more complex interaction schemes, involving multiple body parts and a higher level of detail for a broader range of recognizable gestures. For example, the recognition of the entire skeleton allows interaction with objects outside of the HoloLens’ field of view and interaction with both hands at the same time. Moreover, interaction is not restricted to gestures performed with hands but can be extended to other body parts. The skeletal information, in combination with the spatial awareness of the HoloLens, allows for inferring contextual information from natural body movements.

We propose an approach to integrate an SBE-side tracking device with a user-side MR headset to form a “multiple-device interaction proxy” that can extend the embodied interaction in mixed-Reality environments [Handosa et al., 2018]. We rely on the tracking device to capture the body movements and the gestures of the user on behalf of the MR device. Providing such information allows an MR application to overcome the limitations of the MR device and implement more intuitive interaction scenarios. However, before the MR device can benefit from the user tracking information obtained by the tracking device, that information needs be mapped from the tracking device coordinate system to the MR environment coordinate system. Registering two coordinate systems can be achieved by collecting a set of point pairs. Each pair consists of two corresponding points, one from each coordinate system. Once those points are collected, a registration algorithm can be applied to obtain a transformation matrix that maps a point from one coordinate system to the other. Several coordinate registration algorithms have been proposed such as the algorithm in [Besl et al., 1992] and the eight-point algorithm [Hartley, 1997].

A system that integrates MR devices with tracking devices is depicted in Figure 7.10. For each tracking device, there is a server application that collects tracking data and makes it available for the interested clients. The MR application should incorporate two modules (a
tracking client module and a registration module). The tracking client module is responsible for obtaining the tracking data from the tracking server while the registration module is responsible for mapping the obtained tracking data to the coordinate system of the MR device. Following this architecture, MR devices can obtain data from several tracking devices and a tracking device can provide data to several MR devices.

Figure 7.10: Integrating an SBE-side tracking device with a user-side MR headset to form a multiple-device Interaction Proxy.

Some tracking devices can track several persons simultaneously, which may or may not include the user. Consequently, an MR device may receive tracking data for several persons. In that case, the MR device will need to identify which data set belongs to the user if any. Given that an MR device is a head-mounted device, the current location of the device in the MR coordinate system gives a good indication of the current location of the user’s head. Comparing the device location with the registered tracking data can reveal which data set belongs to the user if any.

7.4.2 Implementation

Based on the proposed approach, we have implemented a system that integrates Microsoft HoloLens devices with Microsoft Kinect devices. A Kinect server application tracks the user skeleton using the Kinect device. The HoloLens application obtains the tracking data from
the server through its Kinect client module before the registration module maps it to the HoloLens coordinate system using a transformation matrix.

In order to obtain the transformation matrix, we have developed a four-step process to collect four point-pairs, where each pair consists of a point under the Kinect coordinate system and its corresponding point under the HoloLens coordinate system. Each point pair is collected by asking the user to place a hand at a position in space that is indicated by a hologram as shown in Figure 7.11. Once the user’s hand is in position, the hand tracking information is collected from both the Kinect device and the HoloLens device to form a point pair. After collecting the four point-pairs, an algorithm is applied to obtain the transformation matrix, which can be used later to register the tracking data collected by the Kinect device into the HoloLens coordinate system. Figure 7.12 shows a set of HoloLens-rendered holograms aligned with the tracked joints of the corresponding physical body.

![Figure 7.11: Collecting a point-pair from the Kinect device and the HoloLens device.](image)

In order to communicate the tracking data from the tracking server to the tracking client, we have tested two communication models (direct and indirect) as shown in Figure 7.13. For direct communication, we use the User Datagram Protocol (UDP). The server has a predefined listening port to which clients can send subscription requests. The server collects tracking data from the Kinect device before sending it to all subscribing clients. This com-
communication model minimizes the communication delay. However, for multiple-Kinect setup, a HoloLens will need to communicate with multiple servers. Establishing several connections with different servers complicates the networking model and makes network troubleshooting more challenging.

![Figure 7.13: Two alternative communication models: (a) UDP-based direct communication and (b) MQTT-based indirect communication.](image)

For indirect communication, we use the MQTT protocol, which can support multiple-Kinect/multiple-HoloLens setups while minimizing the complexity of the communication model. The MQTT broker can support indirect communication between the Kinect server applications and the HoloLens client modules. Each Kinect server can publish the collected tracking data to a specific topic on the MQTT broker. Unlike direct communication, a
HoloLens client module will need to maintain only a single connection with the MQTT broker. The client can subscribe to one or more topics to receive tracking data from one or more Kinect servers. Although indirect communication may increase the communication delay, it allows for relaxing the complexity of the communication model.

7.4.3 Evaluation

In order to evaluate our approach in a non-lab environment, we used it in the development of a HoloLens application for Nurse Aide skills training [Gračanin et al., 2018]. The goal of the application is to augment the student’s experience in classroom settings and to provide a rich set of educational contents in an MR environment. The developed HoloLens application recreates the scenery of a hospital room. Within this virtual hospital room (Figure 7.14a) are the required objects and props to perform the skills in a “close to reality” environment. Figure 7.14b demonstrates an embodied interaction with digital entities (a denture and a toothbrush).

Figure 7.14: (a) The virtual hospital room. (b) Demonstration of using both hands simultaneously for brushing a denture.

Almost all skills require at some point a more detailed user tracking than the HoloLens alone can provide. For example, a crucial part of the hand washing skill requires the student to
keep the hands and forearms at a downward angle to prevent “contaminated” water to run down the arms. With the HoloLens alone, there is no possibility to check this condition. Another example is denture brushing where a student should hold a denture in one hand and a toothbrush in the other hand. With HoloLens alone, enabling hand tracking will require the student to maintain both hands in the ready state (Figure 7.9a) and within the HoloLens field of view, resulting in constrained and unnatural interaction. Fortunately, with the additional data about the entire user skeleton provided by the Kinect device, we were able to achieve a level of detail and precision to track the user’s actions sufficiently.

Users should receive instant feedback as they interact with an MR environment. A noticeable delay in response to user commands can degrade the user experience. Therefore, the skeleton information of the user should be delivered to the HoloLens device with minimum latency to ensure the responsiveness of the system. The responsiveness is determined by the delay (latency) between the time at which the user makes a given gesture/move and the time at which the user receives the corresponding feedback through the MR device.

For the purpose of estimating the overall latency, we have captured multiple MR video recordings of user gestures, specifically the closed hand gesture. Exploring the frames of the captured videos revealed that it takes at most four frames for the HoloLens device to provide feedback after the user gesture takes place. Figure 7.15 shows six consecutive frames from a captured MR video (30 frames per second). The user starts with an open hand and the HoloLens displays a red box indicating that the hand state is open (Figure 7.15a). The user starts to close the hand but it is not closed yet (Figure 7.15b). The user’s hand is closed (Figure 7.15c). The user is waiting for the feedback (Figure 7.15d, e). The user receives a visual feedback and a green box is shown (Figure 7.15f).

Assuming that frame b was captured at time 0 and frame c was captured at time $T$, then the user gesture takes place at time $t_1$, $0 < t_1 \leq T$. Similarly, if frame e was captured at time $3T$ and frame f was captured at time $4T$, then the feedback has occurred at time $t_2$, $3T < t_2 \leq 4T$. Consequently, the delay $d$ is $2T < d < 4T$. The video frame rate is 30
Figure 7.15: Video recording consecutive frames for detecting a closed hand gesture using Kinect and providing feedback through HoloLens.

frames per second and hence $T = \frac{1}{30} = 0.03333$ seconds or 33.33 milliseconds. Therefore, the total system delay ranges between 66 and 134 milliseconds. This estimated latency is caused by the processing and communication steps that take place between a change in user skeleton state and providing the corresponding feedback.

The overall latency $d \geq d_1 + d_2 + d_3 + d_4$, where $d_1$ is the time it takes the Kinect device to capture a frame and send its data to the workstation; $d_2$ is the time it takes the workstation to extract skeleton information from the received frame data producing a skeleton information message; $d_3$ is the time needed to send the skeleton information message from the workstation to the HoloLens; and $d_4$ is the time it takes the HoloLens to provide a feedback based on the received skeleton information. Delays $d_1$ and $d_4$ are device specific and beyond our control. The measured average values of $d_2$ and $d_3$ are approximately 0.157 and 0.476 milliseconds, respectively. Compared to the overall latency, both $d_2$ and $d_3$ are negligible.

The HoloLens device can recognize two simple gestures (release and press). A closed fist with the index finger pointing up indicates a release gesture (Figure 7.9a). Moving the index finger down forming a closed fist indicates a press gesture (Figure 7.9b). Meanwhile, the Kinect device can recognize open and closed hand gestures. Benefiting from the similarity...
between the closed hand gesture of the Kinect device and the press gesture of the HoloLens device, we were able to measure the relative latency of the Kinect-based recognition using the HoloLens-based recognition as a reference point.

Although the Kinect-based recognition involves several processing and communication steps, results have shown that its performance is comparable to that of the built-in HoloLens recognition. In fact, Kinect-based recognition can often perform faster than the built-in HoloLens gesture recognition. Figure 7.16 shows a closed hand (or press) gesture, where the Kinect-based recognition performed the HoloLens-based recognition with approximately 51 milliseconds.

![Figure 7.16: A closed hand (or press) gesture recognized by both Kinect and HoloLens.](image)

Using the MQTT-based indirect communication can simplify the communication model and make network troubleshooting less challenging, especially for multiple-Kinect/multiple-HoloLens setups. However, a significant increase in communication latency can degrade the user experience. The MQTT protocol supports three Quality of Service (QoS) levels. The message delivery for QoS-0, QoS-1, and QoS2 are at-most-once, at-least-once, and exactly-once, respectively. Test results have shown that the average delay for the QoS-0, QoS-1, and QoS-2 are approximately 2.743, 28.492, and 36.047 milliseconds, respectively. Although QoS-0 provides the smallest average delay, it allows for message dropping. However, this should
not be a problem for applications that are interested in receiving the most recent tracking sample rather than receiving every tracking sample. Compared to average communication delay of the UDP-based direct communication (0.473 milliseconds), the use of MQTT-based indirect communication with QoS-0 does not introduce a significant delay (2.743 milliseconds) considering that the overall system delay is between 66 and 134 milliseconds.
Chapter 8

Usability Evaluation

Supporting multimodal interaction with SBEs allows for a flexible interaction experience. A user may prefer to use a given UI type over another based on the task at hand or personal preference. Lighting plays a key role in supporting user tasks as well as creating an ambiance in the illuminated space. Therefore, supporting light control tasks in one of the basic requirements for SBEs. This chapter presents a pilot user study that was conducted to compare and evaluate the usability of four different UI types (GUI, voice-based, gesture-based, and MR-based) when used in an SBE to perform four basic light control tasks (turning a light source on, turning a light source off, adjusting the brightness of a light source, and adjusting the color temperature of a light source). The study setup is illustrated in section 8.1 and the study procedure is explained in section 8.2. Afterward, section 8.3 identifies the different types of collected data and describes the data collection methods. section 8.4 describes the subjects who participated in the study. Finally, section 8.5 discusses the study results.
8.1 Study Setup

The study setup uses several devices. As shown in Figure 8.1, five light bulbs (OSRAM Lightify bulbs) are positioned along with a tracking device (Microsoft Kinect) in front of one of the walls in the room where the study is conducted. One light bulb acts as a reference bulb while the other four bulbs can be controlled by the subject. The four controllable light bulbs are given convenient names to make it easier for the subject to distinguish between them (Top Left Light, Top Right Light, Bottom Left Light, and Bottom Right Light). During the study procedure, the subject should be facing the Light bulbs and the tracking device.

![Figure 8.1: Layout of the study equipment.](image)

The subject is provided with four different UI types (GUI, voice-based, gesture-based, and MR-based) to control lights. The GUI is a vendor-provided smartphone app (OSRAM Lightify app) while the voice-based UI is provided via a smart speaker (Google Home) using a vendor-provided integration setup. The design, implementation, and performance of both the smartphone app and the smart speaker are beyond our control. They represent two popular state-of-the-art approaches to providing thing UIs that are widely adopted nowadays by thing vendors. We provide two additional UI types (Gesture-based and MR-based) that we have implemented to explore other interaction modalities. The gesture-based
UI captures gesture inputs using a tracking device (Microsoft Kinect) while the MR-based UI is provided using an MR headset (i.e. a Microsoft HoloLens app). The experiment is guided by a computer application that provides the subject with instructions to perform a set of predefined tasks and collects data about their performance in these tasks.

The study setup, as shown in Figure 8.2, consists of a set of devices and software components that communicate using different protocols. The OSRAM Lightify Gateway lies at the core of the setup. It uses the Zigbee protocol to communicate with the connected light bulbs. The gateway acts as an intermediary to control the lights as well as to obtain information about their current state. The vendor-provided smartphone app and the smart speaker can use a vendor-specific binary protocol to communicate directly with the gateway over a local network using a TCP connection. Alternatively, they can use a vendor-provided cloud service with a RESTful API to communicate remotely with the gateway over the Internet.

The gesture-based and the MR-based UIs communicate with the gateway using the MQTT protocol. The Kinect MQTT Client captures gesture inputs using Microsoft Kinect before sending it to the MQTT broker. Similarly, the Microsoft HoloLens app captures inputs before sending it to the MQTT broker. The MQTT broker, in turn, forwards the messages it receives from the Kinect MQTT Client and/or the Microsoft HoloLens app to the Lightify MQTT client, which translates the MQTT messages into Lightify protocol packets before sending them to the gateway. Finally, the study controller application guides the study procedures by assigning tasks to the subjects and collecting data about their performance. Moreover, it can communicate with the gateway to configure the light bulbs before each task and to collect data about their status after each task.

8.2 Study Procedure

The study requires each subject to attend two sessions (Session 1 and Session 2) conducted on two different days. During the sessions, the subject is asked to perform three rounds of
assignments (Round 1, Round 2, and Round 3). Round 1 takes place in Session 1 while both Round 2 and Round 3 take place in Session 2. Each round consists of 16 assignments, where an assignment requires the subject to perform one of the four light control tasks using one of the four provided UIs. The round presents the 16 assignments in random order and the light bulb to be configured by an assignment is randomly selected as well.

Besides the lighting control assignments, each subject completed a background questionnaire at the beginning of Session 1 as well as a post-session questionnaire after each of the two sessions. The background questionnaire consists of a set of demographic questions as well as questions about the previous experience with the four modalities. For the post-session questionnaire, we used the System Usability Scale (SUS) [Brooke et al., 1996] to evaluate the usability of each of the four UIs.
8.3 Data Collection

Both qualitative and quantitative data were collected. Qualitative data were collected using the background questionnaire (Appendix A) and the post-session questionnaire (Appendix B). Each subject completed the background questionnaire once at the beginning of Session 1 and completed the post-session questionnaire twice (at the end of Session 1 and at the end of Session 2). This allowed for comparing the responses of the subjects in both sessions to identify improvements and/or inconsistencies.

The subjects were asked to complete three rounds, each of 16 assignments. For each assignment, a set of quantitative measures were recorded, which are (1) the start and finish time of the assignment and (2) the state of the light bulbs after completing the assignment.

During the first round, the subject completes each of the 16 assignments for the first time. Consequently, the time it takes the subject to complete the assignment can provide an indicator to the learnability of the UI. The second round takes place in Session 2, which is conducted approximately two weeks after Session 1. Consequently, the time it takes the subject to complete the assignment can provide an indicator to the memorability of the UI. Finally, the third round takes place right after the second round. Consequently, the time it takes the subject to complete the assignment can provide an indicator to the efficiency of the UI.

Each assignment provides the subject with a reference configuration before asking to apply this configuration to specific light bulbs. The difference between the state of those light bulbs after the subject completes the assignment and the provided reference configuration can provide an indicator of the resulting error when using each of the UIs to complete the assignment.
Table 8.1: Demographics of the participating subjects.

<table>
<thead>
<tr>
<th>Age</th>
<th>Gender</th>
<th>Language</th>
<th>Education Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>21-29</td>
<td>5</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>30-39</td>
<td>5</td>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 8.2: Experience of the participating subjects.

<table>
<thead>
<tr>
<th></th>
<th>Smartphone</th>
<th>Smart Speaker</th>
<th>Kinect</th>
<th>HoloLens</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used it before</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>0</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>To control lights</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>To control other devices</td>
<td>5</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

8.4 Study Subjects

The study required the subjects to be at least 18 years old at the time of participation. A group of 10 subjects has participated in this study. The demographic information of the participating subjects is summarized in Table 8.1 and their previous experience with the study assignments and the UIs in use are summarized in Table 8.2.

8.5 Results and Discussion

This section analyzes the data collected during the study and discusses the results obtained from both quantitative data analysis and qualitative data analysis.
8.5.1 Quantitative Data Analysis

The average completion time for each of the 16 assignments in round 1 is shown in Figure 8.3. The figure shows the average completion time for each of the four light control tasks using each of the four UIs. Results show that subjects were able to perform more efficiently using the GUI compared to the other UIs in three out of the four light control tasks (turn on, turn off, and set brightness) while the gesture-based UI performed better in the remaining task (i.e., set color temperature). Meanwhile, voice-based and MR-based UIs did not perform as well. This is expected as users have experience with smartphones, which helped them to learn the GUI faster than the other UIs. Subjects in this round have used the UIs for the first time to accomplish the assigned tasks. Therefore, the average completion time of a given task using a given UI can provide an indicator of the learnability of that UI.

![Figure 8.3: Average task completion time in round 1 (learnability).](image)

The average completion time for each of the 16 assignments in round 2 is shown in Figure 8.4. The figure shows the average completion time for each of the four light control tasks using each of the four UIs. Results show that subjects were able to perform more efficiently using the gesture-based UI compared to the other UIs in three out of the four light control tasks (turn on, set brightness, and set color temperature) while the GUI performed better in the remaining task (i.e., turn off). Meanwhile, voice-based and MR-based UIs did not perform
as well. Subjects in this round have used the UIs after several days from the first time they used them. Therefore, the average completion time of a given task using a given UI can provide an indicator of the memorability of that UI. Results show that the gesture-based UI has better memorability compared to the other UIs while the voice-based UI had the lowest memorability.

The average completion time for each of the 16 assignments in round 3 is shown in Figure 8.5. The figure shows the average completion time for each of the four light control tasks using each of the four UIs. Results show that subjects were able to perform more efficiently using the gesture-based UI compared to the other UIs in three out of the four light control tasks (turn off, set brightness, and set color temperature) while the GUI performed better in the remaining task (i.e. turn on). Meanwhile, voice-based and MR-based UIs did not perform as well. Subjects in this round have used the UIs right after they used them in Round 2. Therefore, the average completion time of a given task using a given UI can provide an indicator of the efficiency of that UI. Results show that the gesture-based UI has better efficiency compared to the other UIs while the voice-based UI had the lowest efficiency.

Although the above results are preliminary and obtained from a small sample (i.e. 10 subjects), they still show that gesture-based UI can be promising for use in light control and
that we should consider exploring other interaction modalities besides the widely adopted GUI and voice-based UIs.

8.5.2 Qualitative Data Analysis

Using the 10 questions of the System Usability Scale (SUS) [Brooke et al., 1996], subjects were asked to fill out a post-session questionnaire describing their experience with the different UIs at the end of both Session 1 and Session 2. Subject responses were given numeric values from 0-4 based on their selection. The sum of these values is then scaled to obtain a final score out of one hundred. The average usability score for each of the UIs in both sessions is shown in Table 8.3. Although the GUI and the voice-based UI have received the highest average scores in Session 1, their average scores have decreased in Session 2. On the other hand, the average score for the gesture-based UI has improved significantly in Session 2 compared to Session 1. This can indicate that the gesture-based UI is a promising interaction modality for controlling lights that can quickly gain wide acceptance. The gesture-based UI did not perform well in the first session compared to the commonly used GUI and voice-based UI. However, as the subjects got used to it they gave it a higher usability score in the second session.
Table 8.3: The average SUS scores for each of the UIs in both sessions.

<table>
<thead>
<tr>
<th></th>
<th>GUI</th>
<th>Voice-based UI</th>
<th>Gesture-based UI</th>
<th>MR-based UI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Session 1</strong></td>
<td>86.25</td>
<td>75.25</td>
<td>64.00</td>
<td>50.00</td>
</tr>
<tr>
<td><strong>Session 2</strong></td>
<td>81.75</td>
<td>68.00</td>
<td>74.75</td>
<td>51.25</td>
</tr>
</tbody>
</table>

The box plots of the obtained SUS scores in *Session 1* and *Session 2* are shown in Figure 8.6 and Figure 8.7, respectively. Comparing both box plots, we can see how the SUS scores for the gesture-based UI have improved in the second session. The outlier in Figure 8.7 is a low score (30) given by one of the subjects to the GUI. According to that subject, the UI design of the app was somewhat complex to understand at first but later the usage experience was fine. The subject also mentioned that despite being an avid user of smartphone apps, help was still needed to even understand the app navigation design.

![Figure 8.6: SUS scores box plot of the UIs for Session 1.](image-url)
Figure 8.7: SUS scores box plot of the UIs for Session 2.

The results of the one-way Analysis of Variance (ANOVA) test are shown in Table 8.4. The p-value in both sessions is less than 0.05, which leads to rejecting the null hypothesis that all means are equal and confirms that the variability in usability between the four UIs is statistically significant. The $R^2$ values for Session 1 and Session 2 show that the percentage of variation in the response that is explained by the model is 29.51% and 25.82%, respectively.

Table 8.4: The analysis of variance (ANOVA) results for the UIs in both sessions.

<table>
<thead>
<tr>
<th></th>
<th>F-value</th>
<th>P-value</th>
<th>$R^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Session 1</strong></td>
<td>5.02</td>
<td>0.005</td>
<td>29.51%</td>
</tr>
<tr>
<td><strong>Session 2</strong></td>
<td>4.64</td>
<td>0.007</td>
<td>25.82%</td>
</tr>
</tbody>
</table>

Besides the SUS scores collected using the post-session questionnaire, we received comments from the subjects on the four UIs, which we briefly summarize below.
• **GUI.** Some subjects mentioned that they found the smartphone app to be convenient and easy to use as they are familiar with smartphones and GUIs. However, some subjects complained about the UI design of the smartphone app and stated that it was difficult to find the brightness/color controls. Some subjects mentioned that the mapping of the GUI controls to the real world devices was sometimes difficult. One suggestion was to include a home map and add the controls on that map at the corresponding locations of the actual lights.

• **Voice-based UI.** Non-native English speakers had difficulties. Sometimes Google Home would not understand the accent or would not accept prolonged pauses or filler words in the command. When the voice command is more complex than just ‘Turn On’/’Turn Off’, some subjects had trouble to follow the specific format of syntax and maintain the pause between words. Forty percent of the subjects were fluent English speakers. Some subjects found this one to be the most convenient as a hands-free, easy and fast option. Some subjects mentioned that a limited number of fixed nominal values for color can be a shortcoming of the system.

• **Gesture-based UI.** Some subjects preferred the hand gestures because it does not require them to carry an additional device. Some subjects mentioned that there is a chance of fatigue in hands. Some subjects mentioned that there is more chance of false-positives in an everyday scenario in case the system falsely reads movements like ‘catch’ to be ‘grab’ gestures. In case the user forgets to release, the light state might change unintentionally because of hand movements. Some subjects suggested that ’Turn On’/’Turn Off’ requires their own gestures as the most frequent commands.

• **MR-based UI.** Some subjects had trouble performing the air-tap gesture of the Microsoft HoloLens. They mentioned that hands got tired while trying to do the tap gesture repeatedly. Some subjects did not like the fact that they had to wear an additional device for a simple task like light control. One major complaint was the bulky headset. Some subjects mentioned that if the headset is developed into a lighter device
like Google Glasses and if it includes additional functionalities like a smartphone, it would be a very good option. Some subjects mentioned that the UI needs a separate on/off button as that is the most used command. Some subjects liked the additional holographic information overlayed on top of the physical devices. The well-defined menu option helped to achieve precision and reduced chance of involuntarily changing the light state.

This study does not necessarily try to prove that a given interaction modality is better than the others but rather suggests that user interaction with SBEs should not be limited to the use of GUI and voice-based UI. Although GUI and voice-based UI are widely accepted nowadays to support user interaction with things, other interaction modalities can be promising and need to be explored. Humans should be able to leverage the interaction capabilities of the whole body, which can be achieved by supporting multimodal interaction and exploring different interaction modalities.
Chapter 9

Conclusion

Embedding sensors and actuators into everyday objects and incorporating those objects into the Internet realizes the IoT vision, where the digital and the physical worlds are integrated and can interact autonomously. This comes with new opportunities as well as challenges that can reform our world and change the way we experience it. Among several application domains of the IoT, built environments can be turned into smart ones that can autonomously collect and apply knowledge in order to improve the user experience. An SBE provides its users with new and/or enhanced functionalities that can facilitate their everyday tasks. However, a key challenge for SBEs to gain popularity is their support for user interaction. No matter how good are the provided functionalities, user interaction support remains a crucial factor for the success of an SBE.

Users may interact with an SBE either implicitly or explicitly. Supporting explicit user interaction requires the SBE to provide a UI through which the users can interact with the things it incorporates and the services it provides. Thing vendors and SBE designers usually rely on mobile apps and web-based portals to provide UIs for their products. This approach does not scale well. As the number of things and SBEs continues to increase the number of the corresponding mobile apps and/or web-based portals will increase as well. Overwhelming SBE users with numerous thing-specific and/or SBE-specific UI implementations can degrade
the user experience.

Relying solely on traditional HCI technologies (e.g. GUI and touch input) to support user interaction in SBEs may not be always feasible. Claiming that such technologies are well-established and almost ubiquitous may be valid but not enough to justify adopting them as the dominant user interaction techniques in SBEs. In contrast to the digital world, the physical world is a different context with different characteristics and constraints. Therefore, supporting user interaction in SBEs may require adapting the current interaction techniques or adopting new ones.

The main objective of this research is to support explicit user interaction in SBEs while minimizing the interaction overhead at the user side. This is achieved by focusing on user activities and providing multimodal user interfaces. We introduce the “Interaction Proxy” concept, discuss its form dimensions, and explain how it fits in a framework that describes explicit user interaction in SBEs. In order to minimize the interaction overhead at the user side, we propose a framework for autonomous UI generation. The framework is technology agnostic with no assumptions about the UI type or the communication protocol in use. It allows for providing usable UIs that are built on-the-fly leveraging autonomously collected information about the SBE, the current context, and the available I/O capabilities.

This research benefits from an IoT testbed developed around an interdisciplinary research project, FutureHAUS. The testbed allows for creating and connecting various things with embedded sensors and actuators, collect data, explore patterns of use and test different alternatives for providing UIs. The FutureHAUS things include appliances, lights, furniture elements, floors, etc. Various presentation medium, from tablets to multi-touch displays and MR devices can be used as interaction proxies to support user interaction with the incorporated things. An example user study has shown that besides the widely used interaction modalities (i.e. GUI and voice-based UI), other interaction modalities for supporting user interaction with SBEs can be promising and should be explored.
Bibliography


12th annual international conference on Mobile systems, applications, and services, pages 301–314. ACM.


Appendix A

Background Questionnaire

We designed a background questionnaire for the usability evaluation user study that is described in chapter 8. The background questionnaire constitutes of two parts, which are (1) questions about demographic information and (2) background questions to learn about the subject’s previous experience of the study equipment and/or tasks. This appendix contains the list of questions that were presented to the subjects in the background questionnaire.
I hereby acknowledge that I am at least 18 years old and that I am not a student of the investigators.

☐ I consent
☐ I do not consent

Subject code (ask the investigator)

How old are you?

What is your gender?

☐ Male
☐ Female
☐ Other (please specify) [ ]
☐ Do not wish to provide

What is your native tongue?

☐ English
☐ Other (please specify) [ ]
☐ Do not wish to provide
What is the highest level of school you have completed or the highest degree you have received?

- Less than high school degree
- High school degree or equivalent (e.g., GED)
- Some college but no degree
- Associate degree
- Bachelor degree
- Graduate degree
- Do not wish to provide

If applicable, what is your major of study?

[Blank]

Are you White, Black or African-American, American Indian or Alaskan Native, Asian, Native Hawaiian or other Pacific Islander, or some other race?

- White
- Black or African-American
- American Indian or Alaskan Native
- Asian
- Native Hawaiian or other Pacific Islander

- From multiple races
- Some other race (please specify) [Blank]
- Do not wish to provide
Are you now married, widowed, divorced, separated, or never married?

- Married
- Widowed
- Divorced
- Separated
- Never married
- Do not wish to provide

Which of the following categories best describes your employment status?

- Employed, working 1-39 hours per week
- Employed, working 40 or more hours per week
- Not employed, looking for work
- Not employed, NOT looking for work
- Retired
- Disabled, not able to work
- Do not wish to provide

Have you used a smartphone before?

- Yes
- No

Please select all that apply to your experience with using a smartphone.

- I have used it to control lights.
- I have used it to control other device (e.g. appliances).
Have you used Google Home or Amazon Alexa before?
☐ Yes
☐ No

Please select all that apply to your experience with using Google Home / Amazon Alexa.
☐ I have used it to control lights.
☐ I have used it to control other device (e.g. appliances).

Have you used Microsoft Kinect Before?
☐ Yes
☐ No

Please select all that apply to your experience with using Microsoft Kinect.
☐ I have used it to control lights.
☐ I have used it to control other device (e.g. appliances).

Have you used Microsoft HoloLens Before?
☐ Yes
☐ No

Please select all that apply to your experience with using Microsoft HoloLens.
☐ I have used it to control lights.
☐ I have used it to control other device (e.g. appliances).
Appendix B

Post-Session Questionnaire

We use System Usability Scale (SUS) [Brooke et al., 1996] to create a post-session questionnaire for the usability evaluation user study that is described in chapter 8. SUS is a widely used tool for measuring the usability of a variety of products and services. It consists of a 10 item questionnaire, each with five response options ranging from Strongly agree to Strongly disagree. This appendix lists the ten SUS questions that were presented to the subjects in the post-session questionnaire.
Subject code (ask the investigator)

I think that I would like to use this system frequently.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Somewhat disagree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- Smartphone app
- Voice commands
- Hand Gestures
- Mixed reality

I found the system unnecessarily complex.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Somewhat disagree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- Smartphone app
- Voice commands
- Hand Gestures
- Mixed reality
I thought the system was easy to use.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Somewhat disagree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- Smartphone app
- Voice commands
- Hand Gestures
- Mixed reality

I think that I would need the support of a technical person to be able to use this system.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Somewhat disagree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- Smartphone app
- Voice commands
- Hand Gestures
- Mixed reality
I found the various functions in this system were well integrated.

<table>
<thead>
<tr>
<th></th>
<th>Strongly disagree</th>
<th>Somewhat disagree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smartphone app</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voice commands</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hand Gestures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mixed reality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I thought there was too much inconsistency in this system.

<table>
<thead>
<tr>
<th></th>
<th>Strongly disagree</th>
<th>Somewhat disagree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smartphone app</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Voice commands</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hand Gestures</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mixed reality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
I would imagine that most people would learn to use this system very quickly.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Somewhat disagree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Smartphone app

Voice commands

Hand Gestures

Mixed reality

I found the system very cumbersome to use.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Somewhat disagree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Smartphone app

Voice commands

Hand Gestures

Mixed reality
I felt very confident using the system.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Somewhat disagree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- Smartphone app
- Voice commands
- Hand Gestures
- Mixed reality

I needed to learn a lot of things before I could get going with this system.

<table>
<thead>
<tr>
<th>Strongly disagree</th>
<th>Somewhat disagree</th>
<th>Neither agree nor disagree</th>
<th>Somewhat agree</th>
<th>Strongly agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

- Smartphone app
- Voice commands
- Hand Gestures
- Mixed reality