Task T1306A, Report

“User programming in M3”

Mike Holenderski, Tanir Ozcelebi

Eindhoven University of Technology

1 Introduction

1.1 Goal

The goal of this task is to investigate how to translate lighting preferences provided by the user, or inferred by the smart space, into policies in RDF format, so that they can be utilized by Knowledge Processor nodes in our M3 enabled SmaCS lighting testbed.

1.2 SmaCS carrier project

The aim of the SmaCS project is to create critical software building blocks for providing contextual services to people in various environments, and use cases which go beyond the traditional view and implementations of context awareness for services. The project addresses critical questions such as how to arrive at a more holistic view of requirements for the service web and human-systems-devices interaction, how to address and mine the complexity of the context data in practical cases and how to translate these new embedded intelligence design patterns into business models.

1.3 Our contribution to SmaCS

Research over the years has shown that light has significant influence on various aspects of human life including social and health effects. This creates the need for developing efficient lighting at home, workplace etc. But it is also evident that the preference of light varies over people, activities, mood and many other factors. This calls for the development of intelligent lighting solutions to provide varied lighting conditions based on these factors.

1.4 SmaCS testbed

The SmaCS testbed is based on the office scenario. In particular, it is based on the breakout area, which the office employees use for informal meetings, recreation etc. The goal of this pilot implementation is to develop, test and demonstrate innovative smart space applications and provide the users with pleasant experience through various lighting conditions based on their preferences communicated by means of innovative user interfaces.

Figure 1 below shows the floor plan of the pilot breakout area. The area is divided into two spaces dedicated for different purposes: meeting area for informal meetings and the retreat area for personal retreat and relaxation. The current implementation at the breakout area contains numerous lighting elements that can communicate together. The lighting provides a platform that can be used for various applications such as task lighting, atmospheric lighting or informative lighting.

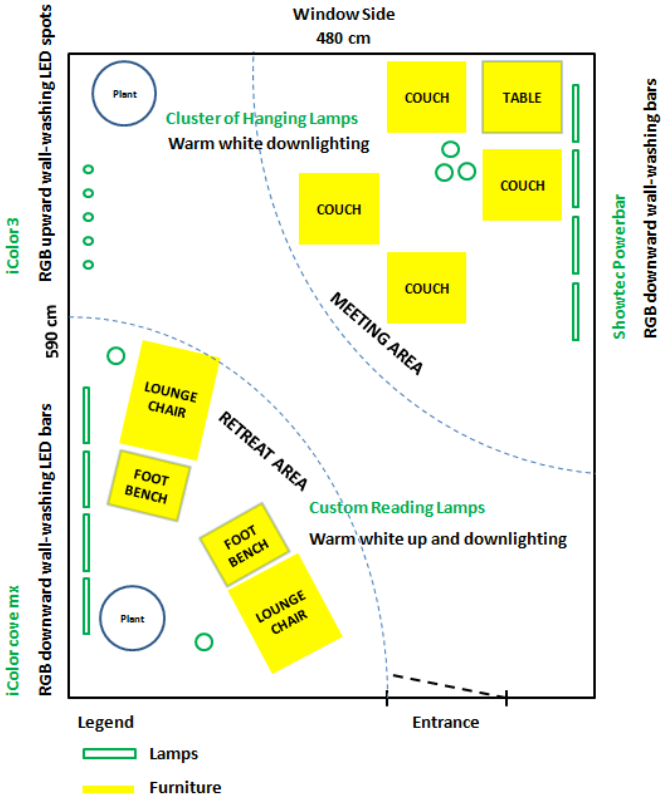

Figure 1: Floor plan of the SmaCS testbed.

1.5 M3

M3 provides a platform allowing the integration of devices from various vendors on the semantic level. It consists of Knowledge Processors (KP) and a Semantic Information Broker (SIB). The SIB stores and communicates information received from KPs in RDF triples format. It allows the KPs to subscribe for certain information and will notify them when this information is published or modified. The KPs are deployed on top of the devices and provide an abstract software interface to the devices' functionalities. There can be virtual KPs, which generate new information from the information already present in the smart space, or which act as a gateway between different smart spaces. The M3 architecture can therefore be regarded as an active black board where data is stored and communicated in the RDF triples format.

2 User programming in M3

Programming a smart space can be regarded as extending it with new functionality that can be used by the devices in the smart space. In this task we propose a method for allowing a user to extend a smart lighting space by defining lighting policies. Once the policies are defined, the user can enact them on the smart space using simple devices. The policies can also be used by other KPs, such as autonomous lighting control, mapping sensory input to the desired lighting policy. The specific light settings are defined in the policy.

The user interacts with the M3 enabled smart space via a mobile app. The app allows the user to define a mapping between a lighting policy and the particular light settings. The current app targets iOS, and can be deployed on iPhones, iPods and iPads. The user interface is shown in Figure 2.

Figure 2: The app user interface allowing to assign individual light settings to a light policy.

After connecting to a smart space, the app retrieves the lighting policies defined in the space. When a lighting policy is selected, the app retrieves the corresponding light settings. The settings for each light in the smart space are shown on a separate screen. In Figure 2 there are four screens (indicated by the four dots) corresponding to the four lights in the smart space for the “Work” light policy. Different light types

can have different settings (e.g. a “Brightness light” has different settings than a “Discrete light”). When the “Set” button is clicked, the light settings are assigned to the selected lighting policy and the policy is enacted on the smart space.

2.1 Architecture

Figure 3 shows the proposed system architecture for an M3 enabled smart space with smart lighting.

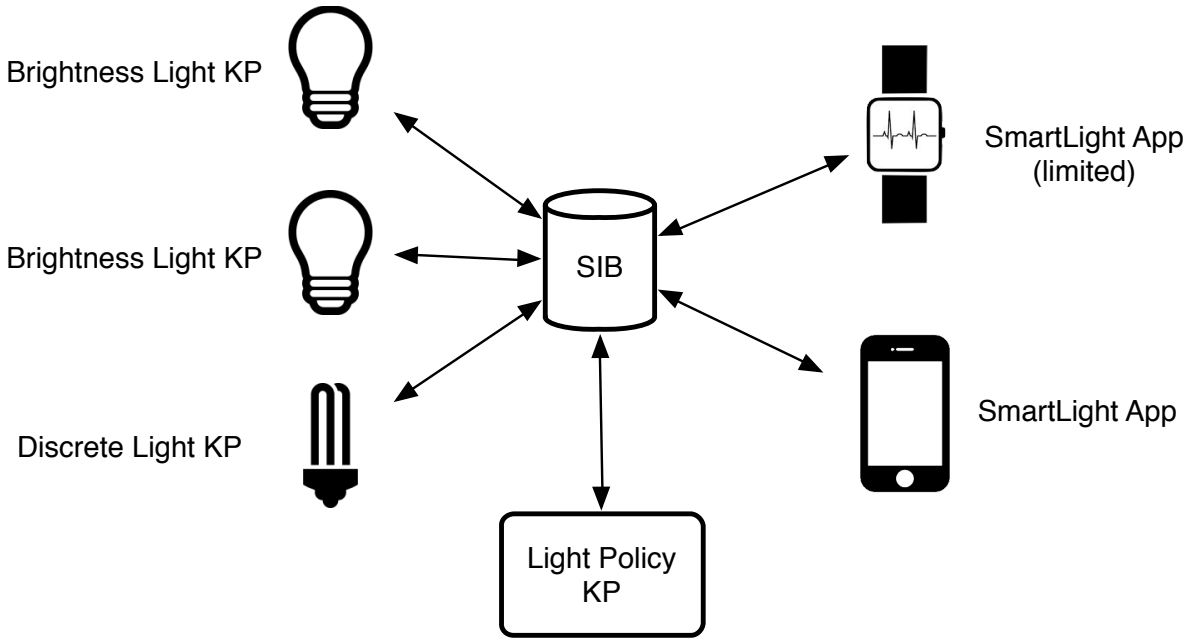

Figure 3: Proposed system architecture

The loose coupling of KPs in a smart space allows extending an existing smart space with higher abstraction level KPs, which provide knowledge derived from other existing knowledge. The Light Policy KP is a good example. It adds the entire light policy management layer to the smart space allowing to build new services on top of it (e.g., the policy control app), without affecting the existing functionality in the smart space.

Once a light policy is defined, it can be enacted on the smart space by a simple device limited capabilities. For example, the small screen in a smartwatch makes it difficult to define lighting policies. However, once policies are defined using his smartphone or tablet, the user can use a smartwatch to select one of the defined policies.

2.2 Ontology

In this section we propose the M3 ontology used to describe the state of a smart space equipped with smart lighting. We use <uri> to refer to an arbitrary URI and <literal> to refer to an arbitrary literal string.

2.2.1 Smart space ontology

A smart space is identified by a literal:

<spaceId> = <literal>

A user of smart space is identified by a URI:

<userId> = <uri>

A smart space has users. The smart space can keep track of currently active users, or users that have used the space in the past. For a user with URI <userUri> the smart space with id <spaceId>

<spaceId> **hasUser** <userUri>

2.2.2 Light ontology

2.2.2.1 Light type

Light type defines the type of a light, and therefore the properties that one can set. Currently we have defined the following light types:

<lightType> = **discrete** | **brightness** | **hue**

2.2.2.2 Light setting

A light with URI <lightUri> is defined by the following triples:

- A light has a name, which can be displayed to the user

  <lightUri> **hasName** <literal>

- A light has a particular type

  <lightUri> **isOfType** <lightType>

Moreover, the current state of a light is defined by a set of triples, one for each of its settings. The settings depend on the light type.

2.2.2.3 Brightness light settings

A light with URI <lightUri> which is of type **brightness** has the following settings:

- <lightUri> **hasBrightness** <brightness>

where <brightness> is a floating point number between 0 and 1 (0 representing dark and 1 representing bright).

2.2.2.4 Hue light settings

A light with URI <lightUri> which is of type **hue** has the following settings:

- <lightUri> **hasHue** <hue>

- <lightUri> **hasSaturation** <saturation>
- <lightUri> **hasBrightness** <brightness>

where <hue>, <saturation> and <brightness> are floating point numbers between 0 and 1.

2.2.2.5 Discrete light settings

A light with URI <lightUri> which is of type **discrete** has the following settings:

- <lightUri> **isWarm** <warm>
- <lightUri> **isBright** <bright>
- <lightUri> **isDynamic** <dynamic>

where <warm>, <bright> and <dynamic> are Boolean values. When <dynamic> is true, then the light color slowly oscillates within a predefined range for the particular <warm> and <bright> combination. When <dynamic> is false, the light color is static.

2.2.3 Light policy ontology

A light policy is a collection of settings for individual lights or lights of a certain light type. A smart space can have several light policies. For a policy with URI <policyUri> the smart space with id <spaceId> has the following triple:

<spaceId> **hasLightPolicy** <policyUri>

A light policy with URI <policyUri> is defined by the following triples:

- A policy has a name, which can be displayed to the user

  <policyUri> **hasName** <literal>
- A policy is bound to a specific user

  <policyUri> **isForUser** <userUri>

A user with URI <userUri> may try to enact a light policy with URI <policyUri> by inserting the following triple into the space:

- <userUri> **wantsLightPolicy** <policyUri>

If there are several users trying to enact their own policy on the smart space, then it is up to the smart space to resolve the conflict.

2.2.4 Light policy setting

A light policy is a collection of light policy settings. For a policy setting with URI <settingUri> the policy with URI <policyId> has the following triple:

<policyUri> **hasLightSetting** <settingUri>

A light policy setting with URI <settingUri> is defined by the following triples:

- A setting is for lights of a particular light type

  <settingUri> **isOfType** <lightType>
- (optional) A setting can apply to a particular light

<settingUri> **isForLight** <lightUri>

If **isForLight** is not specified for a setting, then the setting applies to all lights of type specified by **isOfType**.

Depending on its type, a setting is further defined by the same predicates used to specify the state of a light. E.g., a setting of type **discrete**, is defined by predicates **isWarm**, **isBright**, **isDynamic**.

2.3 Contributions

- We have created a VMWare virtual machine which contains an Ubuntu 10.04 server with a preinstalled SIB, allowing to quickly jumpstart with M3 development.
- We have developed KP libraries for iOS and Tcl, which provide basic M3 communication primitives simplifying the development of application specific KPs. We have used these to implement our SmartLight iOS app, as well as the Tcl based the light policy and virtual light KPs.
- We have developed the SmartLight application (for iPhone with iOS 6.0 or higher) for defining lighting policies in an M3 enabled smart space, which can then be enacted by the app itself or via a device with limited UI capabilities (e.g. a smartwatch).
- We have developed a Light Policy KP, which extends a smart space with support for lighting policies.
- We have developed virtual lights and the corresponding KPs (for Discrete Light and Hue-Saturation-Brightness Light). They simplify the development of smart lighting applications, avoiding the need for real lighting testbeds, or facilitating rapid prototyping for smart lighting applications.

3 Evaluation

3.1 M3

- The fundamental idea behind M3 is well suited for developing smart space applications, i.e. using a central active black board in a smart space to exchange data in the RDF triple format.
- M3 only supports simple queries, allowing to retrieve triples matching a template triple, i.e. a triple with some of its fields missing. More advanced queries, or more "semantic" queries, allowing to specify conditions on the subjects and objects, are not supported. E.g., one may like to query the light policies for a particular user. Currently, a KP must retrieve from the SIB all the policies in a smart space and then filter out those that pertain to a particular user. Instead, the SIB should support specifying a query that will return only policies for a particular user in the first place.
- The RDF graph structure used to store data in a SIB does not support ordered list-like data structures. It only supports unordered sets. This makes it difficult to store information where the notion of a sequence is important.
- What happens when a node leaves the smart space without cleaning up after itself? E.g. when a heart rate monitor fails, the SIB will contain its latest **hasValue** predicate. A timestamp associated with a value could be used to determine when a value is stale.

3.2 Reference SIB implementation

- The current reference implementation of M3 is poorly documented and the documentation that is there is difficult to find on the Internet. The reference implementation of the SIB is distributed in source form only and it depends on several other libraries, for some of the libraries it requires an old version. Getting a SIB compiled and running takes a considerable effort. Also, there does not appear to be an active community around M3. Consequently, in the current state, the reference implementation of M3 is not a viable starting point for developing smart space applications. Without it developers are unlikely to pickup M3 in their projects.
- In the current TCP connector of the reference SIB implementation requires each subscription to go over a different socket. However, there is no fundamental reason for creating and keeping open a separate socket for every subscription. Therefore, there should be an option to multiplex all subscriptions over a single socket between the SIB and a KP.
- The current TCP connector of the reference SIB implementation requires a KP to call the LEAVE method before it leaves the smart space. Otherwise, the SIB does not release subscriptions and resources such as sockets and threads for these devices. Consequently, this leads to the SIB slowing down

and crashing. In a smart space devices may come and go, they may also fail, therefore, robust handling of nodes leaving the smart space is important.