Block 2: Smart Spaces

Title: Study user programming in Smart-M3 (iPhone, iPod, iPad)

Context
A smart space is a physical space enriched with embedded information and communications technology (ICT) and with smart objects that contain ICT. It delivers adaptive context-aware information services, as well as physical services through adaptive actuation. In order to achieve this, a smart space typically combines sensing, wired/wireless communication, computation, cooperation among several objects (e.g. sensors and actuators) and other information (e.g. from the Internet). As opposed to a Wireless Sensor Actuator Network (WSAN), the user is in the center. The main goal is to facilitate execution of adaptive scenarios that are automatically triggered in order to maximize the user experience out of the smart space services and applications. The user not only feels welcomed by the smart space, but he also feels in control through suitable user interaction.

Similarly, nowadays, traditional light sources that just aim to enlighten the space are now slowly being replaced by networked Adaptive Lighting Environments (a particular smart space) that are easily programmable by end users. A very good example is Philips Hue (https://www.meethue.com/en-NL), which, by the way, was initially designed in one of our graduation projects.

Example scenario: Consider a smart home setting that contains a smart chair that can detect presence, a smart television and 4 smart light fixtures:

- 1 smart light fixture \(I_a\) that can emit cool white (CoWhi) light,
- 1 smart light fixture \(I_b\) that can emit warm white (WaWhi) light,
- 2 smart light fixtures \(I_c, I_d\) that can emit warm colored (WaCol) and cool colored (CoCol) light, i.e. they can emit many colors including white.

Consider that all of these objects are wireless devices with TCP/IP connectivity. Initially, the smart home is programmed to behave according to whether or not the user is seated in the smart chair in front of the TV.

<table>
<thead>
<tr>
<th>Before user programming</th>
<th>(I_a)</th>
<th>(I_b)</th>
<th>(I_c)</th>
<th>(I_d)</th>
<th>Smart Chair</th>
<th>TV</th>
</tr>
</thead>
<tbody>
<tr>
<td>User in front of the TV</td>
<td>bright</td>
<td>dim</td>
<td>yellow, dim</td>
<td>orange, dim</td>
<td>occupied</td>
<td>play</td>
</tr>
<tr>
<td>User away from the TV</td>
<td>dim</td>
<td>bright</td>
<td>yellow, bright</td>
<td>orange, bright</td>
<td>Free</td>
<td>pause</td>
</tr>
</tbody>
</table>

The user, User1, with her smart app (e.g. an iPhone), would like to change her customized policies on how this smart home should behave. The new desired settings are as follows.

<table>
<thead>
<tr>
<th>After user programming</th>
<th>(I_a)</th>
<th>(I_b)</th>
<th>(I_c)</th>
<th>(I_d)</th>
<th>Smart Chair</th>
<th>TV</th>
</tr>
</thead>
<tbody>
<tr>
<td>User in front of the TV</td>
<td>bright</td>
<td>dim</td>
<td>blue, dim</td>
<td>blue, dim</td>
<td>occupied</td>
<td>play</td>
</tr>
<tr>
<td>User away from the TV</td>
<td>bright</td>
<td>bright</td>
<td>white, bright</td>
<td>white, bright</td>
<td>Free</td>
<td>Pause</td>
</tr>
</tbody>
</table>
Available:

1) Smart-M3: This is a network architecture in which software entities called Knowledge Processors (KP) can publish (insert/remove/update) their information for other KPs through Semantic Information Brokers (SIB). The SIB stores information received from KPs in RDF triples format. A KP can make queries to a SIB for receiving such information. A SIB allows 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. The M3 architecture can be regarded as an active black board where data is stored and communicated in the RDF triples format. For more details see:

http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5361638

2) A platform for programming in Smart-M3, allowing a user to control KPs (e.g. software on lights) in a smart space: The user interacts with KPs (programs them) via a mobile app.

3) Virtual Smart-M3 KPs representing smart lights.

4) A virtual machine with a working SIB, making it easier to jump start with M3 development.

5) A KP library for Tcl and a build of the KP library for C on the iOS platform.

Information shared in SIB:

The following explains 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.

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.

A. 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>

B. Light ontology

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

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> isOf Type <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.

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

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

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

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

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

- `<lightUri>` is `warm`<warm>
- `<lightUri>` is `bright`<bright>
- `<lightUri>` is `dynamic`<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.

**C. 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>` has `lightPolicy`<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>` has `name`<literal>
- A policy is bound to a specific user: `<policyUri>` is `forUser`<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>` wants `lightPolicy`<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.

**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>` has `lightSetting`<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. For example, a setting of type discrete, is defined by predicates isWarm, isBright, isDynamic.

**Problem Description (Assignment 1)**

Smart-M3 provides only very simple queries. Creating interesting scenarios will require extending it with semantic queries. Such queries should enable a smart space app developer to quickly write a KP which will map certain conditions in a smart space (such as the state of the user or media in the space) to a desired action (such as enacting a predefined lighting policy or adapting the settings of an individual light). It is needed to express conditions on several connected RDF triples, rather than only a single triple. Extend the Smart-M3 querying mechanism to query policies and extend the virtual KP and the SIB such that a user can easily (re)program new desired behaviors such as the one given in the example scenario.

Firstly, the students are expected to extend the Smart-M3 querying mechanism to query a given policy (e.g. all colored lights that are orange and that are bright (brightness=1). Note that a trivial alternative solution would be to make many queries for RDF triples. Secondly, the students are expected to extend the virtual KP such that the results returned by such query can be reprogrammed by just redefining a policy. Note that a trivial alternative solution would be to reprogram each and every KP one by one by inserting values in the RDF tree. The evaluation is through a demonstration of programming the desired behavior and making example policy queries from the smart phone app as well as reporting.

**Problem Description (Assignment 2)**

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. Extend Smart-M3 with support for sequences and illustrate it by defining an interesting dynamic light behavior, e.g.

1. set a light color to blue and set its brightness to 0 (zero).
2. change its brightness from 0 to 1 in a duration of 2 seconds (i.e. it takes 2 seconds to change the state, ignoring communication and processing delays).
3. change its color from blue to red in a duration of 5 seconds.
4. change its brightness from 1 to 0 in a duration of 2 seconds.

The evaluation is through a demonstration of inserting such a sequence as data and the corresponding dynamic light behavior as well as reporting.