Final Project: Smart Campus

Version 0.1	6/12/2017	Michael Lindner	Initial Draft
Version 0.2	14/12/2017	Michael Lindner	Added DB
Version 0.3	21/12/2017	Michael Lindner	Minor Review

Motivation

The SmartHome project was defined to run in a single process. However, we might find out that it should scale in a way that requires more than a single process, or maybe more than a single machine. Moreover, sometimes, a few smart-homes should cooperate to compose a smart-campus. In order to achieve that, we should separate the smart home into separate processes that can communicate with each other.

Process Structure

A campus is composed of the following elements:

- SmartHome - responsible for managing agents and events inside a single building.
  - A campus will run multiple instances of this system one per building.
- Registrar – One instance per campus, this is a central registry for SmartHome instances.
  - Serves for discovery of SmartHome systems
  - Holds the address/port of all the buildings/sections in the campus
  - Holds the information of database connectivity
- DataBase – the DB holds configuration and data
- Reporter – A simple application that allows getting reports about the system

Communication

Communication will be done using a special protocol, SHMP (SmartHome Message Protocol).

1. The protocol is text-based
2. Any message starts by "**`SHMP:`** "
3. After that, comes the command and the possible arguments, separated by space(s)
4. A message ends by a null-terminator
5. The largest message allowed is of 4096 bytes
6. Any message should get a response (text base, null terminated, limited to 4096 bytes)
   a. A response starts with "**`SHMP: status`** "
   b. After that, there is the status code (3 digits)
   c. After that, there is a textual explanation of the status
   d. After that, if relevant, there is a new line char (\n) and the content of the response
7. Any message-request might get one of several possible responses. The following responses are relevant to all the message requests:
   a. **`SHMP: status 200 OK`**
      if that is the expected result (followed by the relevant content, if applicable)

b. **`SHMP: status 400 Bad Request`**
   if the request has syntax error
c. **`SHMP: status 500 Internal Server Error`**
   if the request was legal but there was an error while processing it

Registrar

The Registrar is a special process, whose sole purpose is to provide a single point of contact to all the SmartHome Processes. It listens on a given port (provided from command line argument), using TCP/IP.

The registrar supports the following requests:

- register
- unregister
- get
- list
- tellme

Register a section

Allows a section to register itself. Request syntax is
`SHMP: register <name> <address> subs:<port> evts:<port>`
and must include the name of the registered section, its IP address and the ports on which it listens to both subscriptions and events. On success, the registrar just sends the "200 OK" response with no additional info. In a case that the section is already registered, it will return
`SHMP: status 409 Conflict`

Unregister a section

Allows a section to unregister. Request syntax is
`SHMP: unregister <name>`
On success, the response will be the "200 OK" with no additional info. In a case that the section is not currently registered, the registrar will respond with
`SHMP: status 404 Not Found`

Get a section's details

Allows getting the connection details of a specific section. Request syntax is
`SHMP: get <name>`
On success, the "200 OK" response will be sent, and in its second line, the details of the requested section will be provided:
`SHMP: status 200 OK`
`section <name> <address> subs:<port> evts:<port>`
If the section is not registered, a "404" response is returned:
`SHMP: status 404 Not Found`

Get all sections

Allows getting a list of all the registered sections, with or without their details. Request syntax is
SHMP: list [full]
On success, a "200 OK" response is sent, followed by a line describing the number of registered sections, and then a list of all the sections, one in a line. If the "full" option is provided, then in addition to the name, the list will include the address and ports. For example, this is a possible response to **SHMP: list** :

```
SHMP: status 200 OK
num of sections: 3
section square
section triangle
section round
```

and this is a possible response to **SHMP: list full** :

```
SHMP: status 200 OK
num of sections: 3
section square 10.0.0.2 subs:1212 evts:1213
section triangle 10.0.0.3 subs:1212 evts:1213
section round 10.0.0.4 subs:3333 evts:4444
```

If no section is registered, the num of sections is 0 and nothing follows, e.g.

```
SHMP: status 200 OK
num of sections: 0
```

Get updates on new registrations

If one section would like to be updated when some other section is being (un)registered, it should use a tellme request:

SHMP: tellme requester:<req_name> section:<sec_name>

That will ask the registrar to update the requester when section <sec_name> changes status. If the request is syntactically legal, a "200 OK" should be returned. Then, whenever that section changes status (registers if it was not yet registered, or unregister if it was already registered), the registrar will send a request to <req_name>'s sections subs-port an updating message:

SHMP: update section:<name> now:on

SHMP: update section:<name> now:off

The "now" will be "on" for a newly registered section and "off" to an unregistered message. Upon sending an "update" message, the "tellme" request is discarded. If the requester would like to continue getting updates, it should request a "tellme" again.

Smart Home

Configuration

Each "section" runs a smart-home process. The only configuration required is the section's DB connection details (each section has its dedicated DB server). All the rest is provided in the DB itself. The configuration data includes:

1. The section's name
2. The section's address, port for subscriptions and port for events
3. The registrar's address and port
4. All the agents' data (that were formerly provided in the configuration file)

Registration

Once the section is up and ready, it registers itself to the registrar. Before shutting down, it should unregister itself.

Operation

In general, the section works just like it did in the former (not distributed) version of the smart home; any agent might subscribe to events within the section, either for the whole section, or for specific floors/rooms. Agents might also unsubscribe for a topic. If an agent is already subscribed for a topic and would like to add some more rooms/floors to which it should subscribe, then it just adds a subscription. If it would like to unsubscribe for that topic, then it unsubscribes the whole topic. For the sake of simplicity, if an agent would like to *change* the floors/rooms of a topic to which it is already subscribed (i.e. remove some of them and possibly add others) then it should fully unsubscribe the topic and then re-subscribe with the new floors/rooms.

Cross Section Events

Subscription

An agent might also subscribe to events coming from other sections. Each such subscription is for a specific topic, and might be for

- Any other section
- A specific section
- Specific floor in a specific section
- Specific room in a specific section

In order to subscribe on a given (other) section, one has to connect that section's address/subs-port and send a message request:

```
SHMP: subscribe section:<subscriber> topic:<topic>
```

```
SHMP: subscribe section:<subscriber> topic:<topic> floor:<floor>
```

```
SHMP: subscribe section:<subscriber> topic:<topic> room:<room>
```

For example, if an agent from section "square" would like to subscribe to the topic "security" anywhere on section "round", then section "square" should send the following message to "round":

```
SHMP: subscribe section:square topic:security
```

Or, if an agent from section "round" would like to register to "fire" messages on floors -3 in section "triangle", then "round" should send that message to "triangle":

```
SHMP: subscribe section:round topic:fire floor:-3
```

If the request is fulfilled, then the other section responds with "200 OK" response. Otherwise, error (4xx or 5xx) might be sent.

Unsubscription

In order to stop subscription to another section, an unsubscribe message should be sent to the address/subs-port of the relevant section, e.g.:

```
SHMP: unsubscribe section:<subscriber>
SHMP: unsubscribe section:<subscriber> topic:<topic>
SHMP: unsubscribe section:<subscriber> topic:<topic> floor:<floor>
SHMP: unsubscribe section:<subscriber> topic:<topic> room:<room>
```

Note that it is possible to totally unsubscribe (all topics and all locations) in a single request. Also, pay attention that it is the unsubscriber section's responsibility to make sure that none of its agents is interested in the removed subscriptions. For example, suppose that two agents of "square" are interested in topic "fire" on floor 13 of "round". Now, one of them is no more interested. Sending an "unsubscribe" message will cause the other agent not to get the messages as well.

If the request is syntactically legal, and could have been fulfilled, a "200 OK" status is responded. That is true even if no subscription had to be removed.

Exchanging Events

Sending Events

Events are sent between sections using the evts-port. Events messages comply to the rules of the SHMP protocol. Sending an event is done using the **event** request, in the following way:

```
SHMP: event topic:<topic> type:<type> section:<section> floor:<floor> room:<room>
agent:<sender_id> [<data1>:<value1> <data2>:<value2> …]
```

The event message provides the following data:

- Topic: The topic name
- Type: A unique type ID that allows restoring the event on the receiving machine
- Section, floor, location: The location from which the event is originated
- Agent: ID of the agents that sent the message

Optionally, an event might also include additional info, in the form of additional <key>:<value> pairs.

The response should be one of the 3 basic response (200, 400, 500).

Receiving Events

Upon receiving an event, one of the followings should happen:

- If there is a message factory for the requested type, it will create the event using that factory.
- Otherwise, a default message factory will be used, that includes the mandatory data members, and a map of key:value pairs of the extra data.

After creating the local instance of the event, it is published to all relevant subscribers.

Providing Event Serializer/Deserializer

Any event should implement a serializing function, that allows "translating" its data to string pairs of key:value.

In addition, any third-party SW provider might supply, together with its agent logic in the .so file, functionality for creating a new event from provided strings.

DataBase

Any building has its own database. The database serves two main goals:

1. Provide the configuration for start-up
2. Save data about relevant events

Configuration Data

General

The general configuration includes:

- Registrar address and port
- Self-address and ports (for subscriptions and for events)
- Self-name

Agents

The database will hold a list of all the relevant agents (instead of using the conf-file). For each agent, there are the ID, model type, floor and room. An agent might also have an additional configuration string.

If an agent fails to be initialized three times, it will be marked as failed and the system will no more try to create it (even after restart). In addition, an “alarm” record will be added.

Events

The DB will allow getting the following statistics:

- How many events were sent by each agent
- How many events were sent for each topic
- How many events were sent by each agent/topic

Alarms

The DB will allow querying for the “active alarms”. Each alarm holds a creation data and has a state: NEW, IN_PROGRESS, CLOSED. The system can only open NEW alarms; changing their mode is done by a human intervention. Currently, there are three types of alarms (more might be added).

1. Creation Alarm: Opened when an agent fails to be created.
2. Internal Alarm: Opened when there is an internal error.
3. Connection Alarm: Opened when there is a connection problem.
4. Agents Alarm: Opened by the agents when there is an internal problem.

It should be possible to get the following reports:

1. The number of NEW and IN_PROGRESS alarms on each of the four categories
2. A list of all the NEW or IN_PROGRESS alarms
3. A list of full details of all the NEW or IN_PROGRESS alarms per a given category. The full details include:
    a. Creation: The ID, model type, floor, room and (if exists) config string of the agent
    b. Internal: The name of the module and a description
    c. Connection: The IP and port
    d. Agent: The agent ID, the level of the alarm (FATAL, ERROR, WARNING) and a description

There should be a separate, simple application that allows connecting to a DB and querying those reports.