LISI Core — A Versatile Server Framework for Monitoring and Control of Devices and Services in Smart Home Environments

Jakob Doppler, Christian Gradl

University of Applied Sciences St. Pölten
Institute for Creative\Media/Technologies
St. Pölten, Austria

{jakob.doppler, dm131558}@fhstp.ac.at

Abstract

LISI Core is a modular, Java-based server framework for monitoring and control of heterogeneous devices and services in smart home environments. While many toolkits offer this functionality for a wide range of bus systems and third party hard- and software to the end user they fail at providing simple and affordable means of programming, customization and extension to software engineers and UI designers for rapid-prototyping and building individual solutions.
At first this paper explains the details of the LISI Core architecture, its core components (plug-in mechanism, REST interface, session management). Secondly a range of implemented sample plug-ins and a RESTful tablet client interface are discussed. These currently include temperature, lighting and shading control and energy and presence monitoring via sensors and actuators on a Beckhoff Twincat embedded server, TV & entertainment via XMBC media player API and an audiovisual information display for real time energy feedback.
Finally the field test experiences of the server framework running in a real smart home environment are discussed. LISI Core was tested in the LISI home. LISI is the innovative, solar powered home of Solar Decathlon Team Austria. Solar Decathlon is a biannual competition hosted by the US Department of Energy to promote the application of solar technologies in buildings. Twenty homes designed and built by university teams were open to the public and evaluated by a jury in October 2013 in Irvine, California. Among all twenty preselected international teams to compete, LISI was the overall winner.

1 Introduction

1.1 Requirements

As part of the solar-powered smart home LISI (Living Inspired by Sustainable Innovation)[1] a home automation system was required to address the basic needs in future living, work and leisure scenarios such as (semi)automated control of heterogeneous devices and services and live monitoring and logging of all available parameters – all by means of explicit and implicit interaction.

With LISI Core a framework was designed to run a server on a single low-powered machine that meets the following Solar Decathlon 2013 competition rules and building code [1] and the functional requirements of Team Austria's LISI home as much as possible. These include:

- *Energy Balance:* Since LISI is solar-powered energy efficiency is a main goal throughout the building design. For information and communication technology as well as home entertainment a total of only 250 W consumption were estimated. With these components a server machine Lenovo Thinkpad Edge E130 (20 W), an LG LED TV 42LW5590 (80 W), network infrastructure router (5 W), a Netgear PoE switch (10 W), four SFX Technologies Gel Audio Driver GA6 / GA7[2] ceiling-mounted speakers and amplification (50 W), wall-mounted speakers and amplification JBL One (50 W).
- *Affordability:* In order to meet the contest criterion for best affordability and as good as possible the home automation and LISI Core hardware was built using mostly low cost components. The netbook with the additional 128 GB SSD is around € 500. The audio system and TV cost around € 1000 in total. The only optional hardware are the additional ceiling-mounted speakers and amplifier for € 400. After all the home automation system components for metering energy and controlling electrical installations are usually the most expensive position. While dedicated home automation hardware might require a proprietary bus systems and cost around € 5000 and more (see chapter 1.2), the Beckhoff embedded Twincat server in combination with cheap industry standard components is available for € 2000 only.
- *Device and service control* of HVAC via room temperature, dimming and switching of ambient and task LED lighting, north and south shading

1 LISI – house of the Solar Decathlon Team Austria: http://www.solardecathlon.at

2 SFX Technologies Gel Audio Driver GA6: http://www.sfxtechnologies.co.uk/

all via Beckhoff Twincat embedded server, TV, ceiling mounted-speakers, wall-mounted speakers and content (AV playback, ambient information display and sonification) via XMBC running on a connected netbook.

- Real time *monitoring and logging of all control commands and single device energy consumption*.
- Real time *logging, monitoring and feedback methods of person presence areas* using ceiling-mounted radar motion sensors via Beckhoff Twincat embedded server.
- *RESTful Client API* to expose all device and service functionality mentioned above to arbitrary clients anywhere in the local network or Internet.

1.2 Home Automation Middleware – State of the Art

With the advent of cheap, embedded and low-to-none powered sensors, actuators and PC hardware home automation gets affordable for private homes and help turning more and more smart home concepts into reality. However most of the presented integrated solutions scale from expensive hardware based bus systems over highly complex yet powerful middleware software to proprietary solutions that are not open to programmers or are hardly customizable for the individual needs of various demographic groups of potential home owners. Let alone a young and tech-savy target group smart solutions may give new communication means especially to elderly and people with special needs.

Industry and building automation systems such as KNX/EIB and LonWorks offer a wide range of sensors, actuators and processors for classical home control but usually do not interact with consumer electronics such as TVs and AV media devices that are available in every household. Driven by the high impact of of *mobile ecosystems and gaming consoles* in everyday life software companies such as Microsoft, Apple and Google spread out to the living room and battle over media control solutions but without any attempt to foster ecosystem-independent standardization in the future. Simple device discovery and control standards such as UPnP[3] and DLNA[4] may exist for media devices only since a couple of years but fail to follow pace with new media formats and innovations.

3 Universal Plug and Play: http://www.upnp.org/

4 Digital Living Network Alliance: http://www.dlna.org/

The highly flexible, service-oriented *OSGi* middleware [2] was found by an industry consortium to integrate embedded Java technology into a range of devices from industry appliances to cars to household appliances but requires a lot of knowledge even from experienced programmers. Additionally controlling media is not considered so far. The OpenHab – Home Automation Bus[5] is a promising open source attempt to use OSGi for holistic device and service monitoring and control in smart homes. The project runs on small microcontrollers such as the Raspberry Pi includes bindings to various home automation buses and renders functionality to clients but again it does not integrate (streaming) media.

As fast, reliable and interconnected cloud services emerge web technology specifications and APIs such as HTML5 [3] and nodeJS[6] and D3.js[7] for sockets, streaming media playback, asynchronous processes and visualization are clearly to be considered in future smart home scenarios. But current browsers and scripting environments are often restricted and lack flexibility and performance for simultaneously playback, logging and control proprietary and locally connected hardware sensors and actuators. Often third party libraries and custom built wrappers are required for example communication to connected hardware e.g. RS232. Additionally browser support and incompatible formats e.g. for video tag support [4] add significant complexity. As popular modern operating systems deeply integrate web technologies and services in their OS many of these issues however are expected to be resolved in the near future.

On the hardware side, small integrated microcontrollers with operating systems, hardware accelerated (graphic) chips for USB, HDMI and wireless communication standards such as WLAN, NFC become available at almost no price. A famous example is the renowned Raspberry PI[8]. The same applies to a wide range of even middle to lower class smartphones that additionally host a multi-touch display and even dual- and quad-core processors. LISI Core was designed to run on a low-powered netbook and host performance-critical services such as real-time database logging, REST service,

5 OpenHab – OSGi-based Open source home automation project: www.openhab.org/

6 NodeJS – serverside Java Script platform for fast and scalable network applications: http://nodejs.org/

7 D3.js – a JavaScript visualization library for creation and control of dynamic and interactive graphical forms: http://d3js.org/

8 Raspberry Pi – a credit-card-sized single-board computer: http://www.raspberrypi.org/

video and audio playback, visualization and device control via Beckhoff Twincat simultaneously.

2 System Architecture

2.1 System Overview

The backbone of LISI Core is a Java servlet in a GlassFish 3.1 Java EE Application Server[9] instance with Jersey RESTful Web Service[10] support running on a Thinkpad Edge E130 netbook (Windows 7 32-bit, 4 GB RAM, 1.4 GHz Intel Pentium 977 Sandy Bridge processor, 128 GB SSD). The LISI Core architecture is loosely coupled with an embedded Beckhoff Twincat home automation server that is mounted inside the LISI homes electric control cabinet. The 32-bit limitation was necessary since the Twincat 2 PLC control library[11] only supported this operating system at the time of development. Both servers and wirelessly connected client devices such as the control tablets share the same LAN.

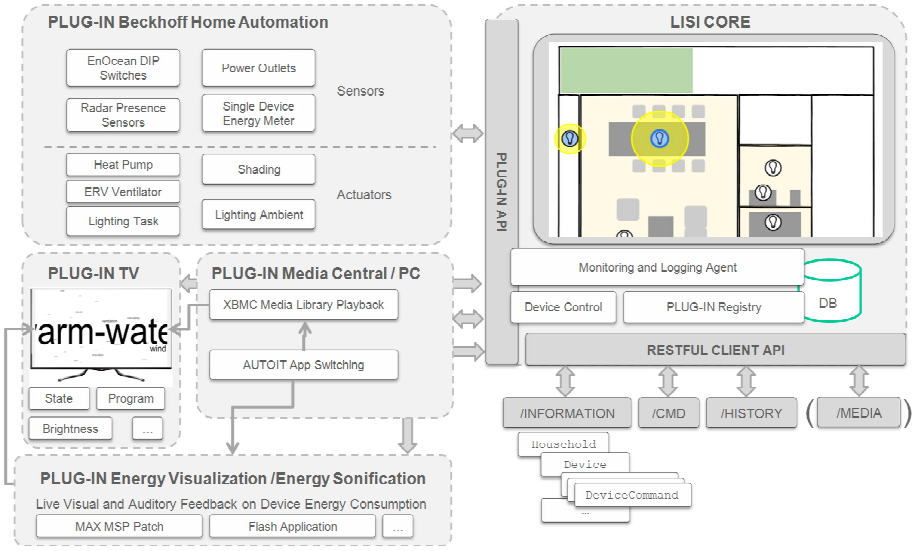

Figure 1 LISI Core – System architecture and various plug-ins (as used in the Solar Decathlon Team Austrias LISI smart home)

9 Glassfish Java EE Application Server: https://glassfish.java.net/

10 Jersey RESTful Web Service: https://jersey.java.net/

11 Twincat 2 PLC control library: http://www.beckhoff.com/english.asp?twincat/libraries.htm

The LISI Core system architecture is outlined in Figure 1. The following components ensure a versatile framework approach for both plug-in developer as well as client developers:
1) Server Specification and Database Abstraction
2) Plug-in Architecture
3) RESTful Client API
4) (Semi) Automated Profiles And Clients

2.2 *Server Specifications and Database Abstraction*

LISI Core is designed as a single servlet within a J2EE server – for the project we rely on a GlassFish 3.1 instance – that uses the server's internal JDBC connection pool to access a local MySQL 5.7 database. Both, the server with servlet and the database are hosted as Windows services that start whenever the OS is booted. Moreover all components of the servlet are hardened against system failure and power blackout that may occur in unknown environments such as the prototypical LISI smart home that is built from scratch in around nine days during the Solar Decathlon competition [1]. This includes: a battery-driven server running on a netbook, a database abstraction layer (DAO) that recovers whenever the connection to the (local) database is lost, a self-healing connection to the Twincat Beckhoff home automation system even if the network is down under house maintenance or due to any unplanned outage.

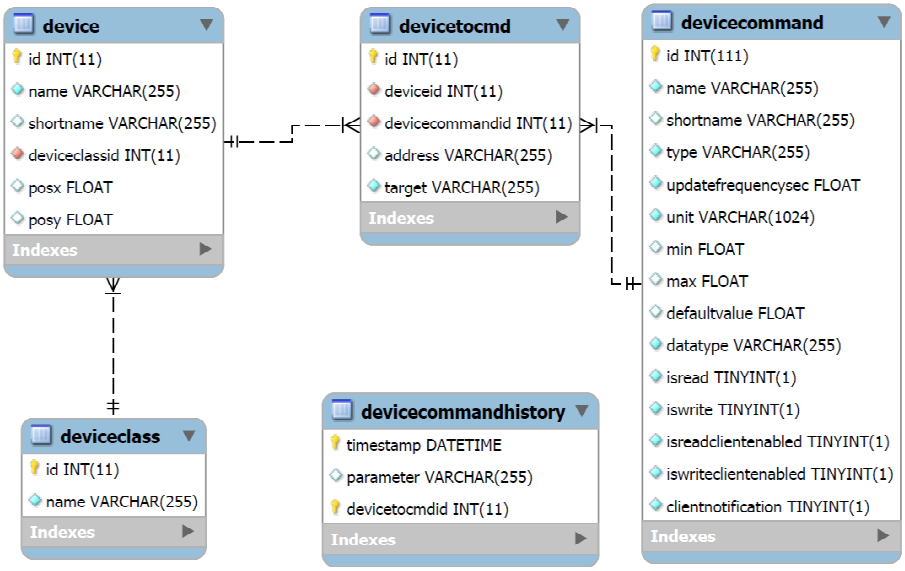

Figure 2 Basic database schema of the device hierarchy within the LISI Core framework

The database schema is designed to support both main functionalities of the LISI Core framework: (a) to load and expose plug-in devices and services easily via a RESTful Client API and (b) to keep all issued commands and state changes in a live session and log them to a history. The log is also available to interested clients applications for visualization purposes. The paradigm of real household devices and appliances and their functionality is kept in the hierarchy of the DB schema (see Figure 2). A `device` is the basic entity – a third-party software or hardware – that might be addressed for control or status request. Every `device` is assigned to one `deviceclass` and has one-to-many `devicecommands`. A `devicecommand` might also be shared by more than one `device` except that it will never have the same plug-in *target* and plug-in *address*. These two combined define a unique id within a plug-in that may serve one or many `devices`.

The following is an example for a household equipped with:

- Eight lights of a `deviceclass` *Dimmable LED* that offer two `devicecommands`: a *dim* command is readable and is writeable with a value between 0 and 100 where zero turns the light off and a readable *energy* command that gives an estimate on the current energy consumption in W.
- Two single lights of a `deviceclass` *Switchable LED* that offers two `devicecommands`: a *switch* command that is readable and writable with 0 and 1 and a readable *energy* command that gives an estimate on the current energy consumption in W.

Depending on the addressed hardware all ten devices might share the same plug-in *target* “Twincat” with different *addresses* “LI100” to “LI111”. During plug-in registration however the framework recognizes that for all devices identical, read-only command energy was found whereas dim and switch are linked only to the corresponding devices.

Besides some more basic parameters the framework defines amongst others: (a) a simple security mechanism that allows commands only to be used within the framework (isread/write) or also on remote clients (isread/writeclientenabled), (b) a DB logging interval with updatefrequencysec and (c) a client notification – that uses a UDP server socket to report command state changes back to clients in a LAN. The latter is used for performance optimization of mobile clients. Since a REST interface requires polling every few seconds the battery life of mobile devices is drained quickly due to costly stateless HTTP calls over mobile Internet and prohibits even a single day of operation in a household. This mode of operation however is still in discus-

sion as the framework might switch to local communication exclusively when the client is in the proximity of the households wireless LAN.

2.3 Plug-in Architecture

Given the database abstraction the versatility of the frameworks roots in its plug-in management for monitoring and control of devices and services. A plug-in consists of any number of Java classes hosting arbitrary HW devices or services. Like in the LISI house these plug-ins might be for example a Beckhoff automation system controlling electric installations and appliances, a TV with a serial interface attached to a laptop hosting an XBMC media server – that is controllable itself via a client API – and third-party application accessing the LISI Core restful client API for live data such as visualizations or sonification via auditory displays [5; pp. 79]. Each plug-in has to provide two basic resources:

- XML configuration file `config.xml`
- Start-up class derived from `IPlugin`

The configuration file helps to setup the plug-ins state. For example a polling rate of an external data source or a IP adress to the remote XBMC media server could be set up. The following block shows an example of the Beckhoff home automation plug-in that is setup using the parameters below and exposes control of one basic light device *LI100* with two commands, namely *dim* and *energy,* via LISI Core the restful Client API (see Figure 1).

```
<plugin target="twincat">
  [...]
  <devices>
    <device name="LI100">
      <longname> TaskLight Kitchen</longname>
      <deviceclassname>Lighting</deviceclassname>
      <devicecommands>
        <cmd address="iLI100dim" name="dim" [...] type="dim"
unit="percent" min="0" max="100" default="0" datatype="int"
isread="1" iswrite="1" iscread="1" iscwrite="1" />
        <cmd address="iLI100energy" name="powercurrent"
[...] type="current" unit="Wh" min="0" max="2000" default="0"
datatype="float" isread="1" iswrite="10" iscread="1"
iscwrite="10"
        [...]
      </devicecommands>
    </device>
    [...]
  </devices>
</plugin>
```

At start-up the plug-in can access all these hierarchically structured <setup> data in the `IConfiBean` object in the startup-class derived from `IPlugin`. Much like and Android `Activity` [6] a few simple states define the lifetime of a plug-in (see Figure 3). `initalizeAndFirstStart` is only called once when the servlet starts. `start` and `stop` could be called multiple times to reconnect broken resources such as a network or a database connection that is down. `destroy` once again is just called right before the servlet is shutdown.

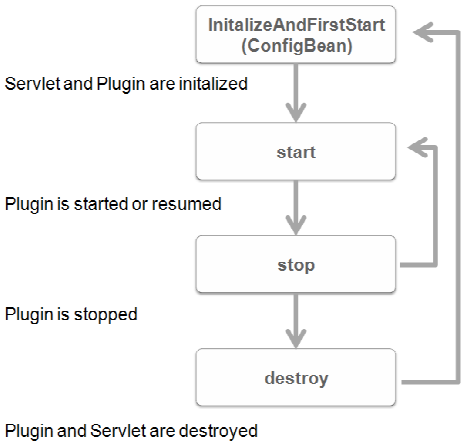

Figure 3 Plug-in lifecycle phases

2.4 RESTful Client API

The main objective of the LISI Core framework is to offer monitoring and control functionality for all available plug-in devices and services to interested clients via a RESTful API (see Figure 1) with special focus on rapid prototyping client applications such as a web interface, tablet or smartphone UI or even information displays or auditory interfaces. Mindful of easy system integration on the client side a flexible household *information* request advertises all homes devices and services in a JSON format. The complete structure of household to devices to devicecommands is derived from a Java object hierarchy using the Google GSON[12] (de)serializer. For Java-based clients such as Android mobiles or desktop applications this even guarantees that the serialized JSON string can be transformed back to a usable OOP household structure. Currently the LISI Core API contains several commands

12 Google GSON: http://code.google.com/p/google-gson/

for reading the household configuration, manipulating states of all devices and reading the history of all device commands:

- `lisiapi/information` – Gets the initial information of the household and all of its devices and their functionality in an easy parseable JSON format.
- `lisiapi/cmd/<deviceid>/<devicecommandid>/[<value>]` – Gets or sets value for a specific device command. E.g. *lisiapi/cmd/LI100/dim/10* sets the dim value to 10, *lisiapi/cmd/LI100/dim* reads the current value.
- `/history/<deviceid>/<devicecommandid>/<intervalstart(YYYYMMddHHmmss)>/<intervalend(YYYYMMddHHmmss)>` – Gets the cmd history of an interval in a JSON format with a 1 min resolution.
- `/media/[<play/picture/pause/stop/addtoplaylist/next/previous/speed>]/[<value>]` – Easily access the media API of a connected XBMC media server. A command without parameters returns all media items in the playlist, the successive commands control the player and playlist behavior. The media request might only be temporarily as it should be included in the default device and device command schema.

2.5 (Semi)Automated Profiles and Clients

With the plug-in management and the RESTful API the LISI Core frameworks helps controlling, monitoring and semi(automating) workflows for clients and internal (semi)automated profiles. The following gives a brief overview of the endeavors in the LISI smart home:

- *Multimedia Scenario Profiles*: We defined four media-assisted household profiles to support everyday work and lifestyles. The Solar Decathlon presentation profiles *Videopresentation* and *MovieNight* turn on the TV, set brightness and soundlevel, play predefined videos and sets shading (in the future depending on light sensors). The dinner patch plays a predefined playlist while dimming lights accordingly for a friendly and low-light evening atmosphere within the LISI home.
- *Presence Activation Profiles*: With invisible cone shaped radar detectors in the intermediate ceiling local lighting and single device-energy profiles can be controlled depending on person presence.
- *Visualization and Sonification*: New metaphors of visual information and auditory displays are employed polling the live parameters such as power consumption on single device level.
- *Client-Control and Information Tablet*: To test and evaluate the use of the LISI Core framework a tablet interface was developed by a team of

the Vienna University of Technology (INSO-DECO). The tablet displays house information such as temperature, humidity, local weather data and energy consumption on single device level. It controls lighting, shading, ventilation, power outlets and media.

3 Future Considerations

Figure 4 Control and Information Display that interacts with the LISI Core framework (done by Vienna University of Technology INSO-DECO, partner of the Solar Decathlon Team Austria)

The system recovered even during maintenance times and power blackouts. At the end of the Solar Decathlon 2013 competition Team Austria was announced the overall winner and industry partner showed interest in marketing LISI Core. Until that however the system will need some more careful considerations in its design and suitability for the market. These include:

- *Extending the (semi) automated profiles:* The concept of lifestyle profiles based on the residents needs could be enhanced with metaphors of implicit interaction and mulit-modal activity recognition within the house.
- *Examine media feedback for related research fields:* With research in smart metering, home automation, digital health care, and ambient Assisted living the use of LISI Core could be extended to address end user

questions such as: How much energy does a single device consume?, how much does it cost?, and What information does an elderly person in a household need to follow throughout the day? Those can be answered by different means of multimedia feedback.

- *System design and security:* While a Windows OS netbook running a Java server might be a convenient and easy to use target platform it is neither very efficient nor very secure. Distributed computing on multiple low powered platforms such as the Raspberry PI with Linux or Android might be an interesting option.

References

[1] US Department of Energy (2013): DOE Solar Decathlon: Rules and Building Code: http://www.solardecathlon.gov/rules.html <2013-11-04>.

[2] OSGi Alliance (2013): OSGi Core Release 5 Specification: http://www.osgi.org/Specifications/HomePage <2013-11-04>.

[3] W3C (2013): HTML5 Editors Draft 7, Dec. 2013: http://www.w3.org/html/wg/drafts/html/CR/ <2013-11-04>.

[4] Wijering, Jeroen (2013): The State of HTML5 Video: http://www.jwplayer.com/html5/ <2013-11-04>.

[5] Kramer, Gregory (ed.) (1994): *Auditory display : sonification, audification, and auditory interfaces.* 1st International Conference on Auditory Display (Santa Fe, N.M.). Reading, Mass.: Addison-Wesley.

[6] Mednieks, Zigurd (2012): *Programming Android*. Sebastopol, CA: O'Reilly.