Colibri: MATLAB Simulink connector

Submitted in fulfillment for the project by
Computational Science and Engineering at TU Wien
under the prestigious program of
Google Summer of Code 2016

Submitted by
Pratyush Virendra Talreja
Indian Institute of Technology Bombay

Under the guidance of
Daniel Schachinger
Thomas Frhwirth

Acknowledgement

I would like to thank Google Summer of Code and team for considering my profile a well deserving for this project. Also, I would like to express my special gratitude to my mentor Mr. Daniel Schachinger, whose contribution in stimulating suggestions and encouragement, helped me to work on my project very well.

I have to appreciate the guidance given by other supervisors that has improved my implementation. I am thankful to my Parents, Teachers and Friends who have been always helping and encouraging me throughout the Summer to work on the project.

I take this opportunity as one of the biggest milestone in my career development. I will work hard to use the gained skills and knowledge in the best possible way, and I will continue to work on the optimization in order to attain desired career objectives. Hope to continue cooperation with all of you in the future.

Thank you.

Contents

List of Figures

Nomenclature

MATLAB	MATrix LABortary
RDF	Resource Description Framework
SPARQL	SPARQL Protocol and RDF Query Language
REST	Representational State Transfer
GUI	Graphical User Interface
INF	Infinite
JSON	JavaScript Object Notation
POJO	Plain Old Java Object
JAR	Java Archive
ODE	Ordinary Differential Equations
TCP	Transmission Control Protocol
XML	Extensible Markup Language
API	Application Program Interface

Chapter 1

Introduction

Colibri provides a platform for smart building energy management. Semantics about the building, the building automation systems, other energy-consuming or energy-producing devices, and the environment are used to elaborate optimization strategies. The decisions are propagated to the devices of the building automation systems in order to influence physical processes within the building.

Often, engineers want to simulate the behavior of building automation systems prior to the realization in a real building. This is also common practice in various other fields of engineering. The individual components can be designed, and their interconnection is simulated in an abstract way using MATLAB Simulink.

1.1 Objectives of the Project

In order to link the simulation with the Colibri platform, a Java-based connector is implemented in this project. This component should be able to read values from and write values to the running MATLAB Simulink simulation. These data exchange, then, needs to be propagated to the Colibri semantic core.

The aim is to write a connector between the semantic core of Colibri and MATLAB Simulink using Java to make the things automated.

Chapter 2

Brief about the Project and the components

To write a connector between the semantic core of Colibri and MATLAB Simulink using Java as programming language. The connector should be able to read and write values from the simulator. The values which are available is specified in the connectors configuration file.

On the other hand, the connector pushes the read data to the Colibri semantic interface using WebSocket communication. Also the semantic core sends values to the connector, which needs to be forwarded to MATLAB Simulink.

The goal is that the connector is able to bridge the gap between the Colibri world and the MATLAB Simulink world.

The project makes use of MatlabSimulink, WebSocket communication and Colibri's Semantic Interface which are described in detail in the following subsections.

2.1 Matlab Simulink

Simulink is an input/output device GUI block diagram simulator. Simulink contains a Library Editor of tools from which we can build input/output devices and continuous and discrete time model simulations.

Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB, en-

abling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

Simulink is a time based software package that is included in Matlab and its main task is to solve Ordinary Differential Equations (ODE) numerically. The need for the numerical solution comes from the fact that there is not an analytical solution for all DE, especially for those that are nonlinear.

What is Simulink good for?

1. Modeling/designing dynamic systems (including nonlinear dynamics)
2. Modeling/designing control systems (including nonlinear controllers and plants)
3. Signal processing design/simulation

In Simulink, systems are drawn on screen as block diagrams. Many elements of block diagrams are available, such as transfer functions, summing junctions, etc., as well as virtual input and output devices such as function generators and oscilloscopes.

Simulink is integrated with MATLAB and data can be easily transfered between the programs. Simulink is supported on Unix, Macintosh, and Windows environments; and is included in the student version of MATLAB for personal computers

2.2 WebSocket communication

WebSockets is an advanced technology that makes it possible to open an interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.

WebSockets represent a long awaited evolution in client/server web technology. They allow a long-held single TCP socket connection to be established between the client and server which allows for bi-directional, full duplex, messages to be instantly distributed with little overhead resulting in a very low latency connection. Figure 2.1 shows the WebSocket Architecture

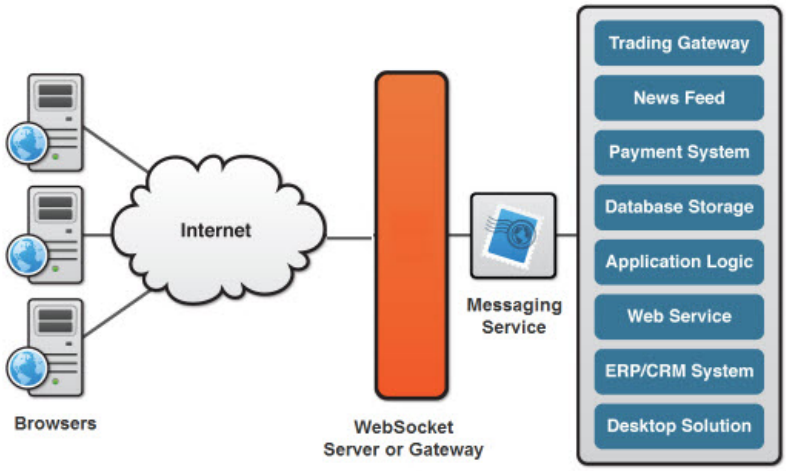

Figure 2.1: WebSocket Architecture

Both the WebSocket API and the WebSocket protocol are standardised which means the web now has an agreed standard for realtime communication between Internet clients and servers. Originally considered a browser technology, WebSockets are reaching far beyond just web browsers and are becoming a cross platform standard for realtime communication between client and server. As well as native WebSocket support in browsers such as Google Chrome, Firefox, Opera and a prototype Silverlight to JavaScript bridge implementation for Internet Explorer, there are now WebSocket library implementations in Objective-C, .NET, Ruby, Java, node.js, ActionScript and many other languages.

2.3 Colibri's Semantic Interface

Colibri provides a platform for smart building energy management. Semantics about the building, the building automation systems, other energy-consuming or energy-producing devices, and the environment is used to elaborate optimization strategies. For this purpose, information is exchanged with the building automation systems as well as external agents, such as smart grid agents or Web service

providers.

In order to prevent the linked systems and components from direct access to the Colibri semantic core, where the information is stored, an interface is implemented in this project. Following best practice in Web service design (e.g. REST), the interface should be able to manage a small set of message types to read, write, and query information. For example, building automation systems push new sensor values towards the Colibri semantic core using a write message, or the Colibri optimization algorithm is able to use the querying functionality to retrieve necessary information.

Both the interface component and the linked components should be able to initiate an information exchange. Thus, the WebSocket protocol is utilized. As a result, the semantic interface will be able to push, for example, a new value for a light switching actuator towards the building automation system. The exchanged information and the queries within the messages are encoded using RDF and the SPARQL protocol, respectively.

2.4 Components

Mainly three components that communicate with each other are used which are:

1. Colibri Test Server
2. Connector
3. MALTAB Simulink

2.4.1 Colibri Test Server

Colibri Test Server opens the WebSocket communication for its clients. To communicate with the running Simulink, there are some message commands specified in the Colibri's specification document that are used here.

2.4.2 Connector

- Connector in the implementation is the ClientEndpoint for the Colibri Test Server i.e. the Connector acts as a client to the Colibri Test Server
- When connector is connected to the Colibri Test Server then it opens the ServerSocket port for MATLAB to connect to it.
- In short, Connector acts as the WebSocket client to Colibri Test Server and Socket server to the MATLAB Simulink

When the connection is made between the components, the connector sends the REG message to Colibri Server and Server responds the Client with STA message. When the services are added then message transfer takes place between Colibri and MATLAB Simulink using Connector as the mediator.

2.4.3 MATLAB Simulink

The MATLAB script is the Socket client for the Connector and it also communicates with the Simulink. The Simulink runs for the INF time and the MATLAB gets the running value from the Simulink using ports when required. The MATLAB script can also send the value to the running Simulink.

The Simulink used here implements the temperature and light services. The simulation used here can send the temperature value when required. Also, light service is used in the simulation with the function that when light is switched on then the temperature is increased and when the light is switched off the temperature is decreased.

Chapter 3

Implementation and GUI

This section mentions the steps to follow for running the project and analyzing the message flow. To run the Colibri: MATLAB Simulink connector:

- Run the Project which will start the Colibri Test Server
- Run the Client.java file for Connector to connect as WebSocket client to the Colibri Test Server and open the Socket port for MATLAB Simulink to connect to it
- Run the .mat file from the Matlab-Simulink folder to connect the MATLAB Simulink to the Connector

Working

- GUI implemented here(1st Screen when we run the Colibri Test Server) is the basic web interface consisting of a TextArea where the incoming and outgoing messages are displayed; a text box where the user needs to enter the message commands and a submit button which submits the user's command to ServerEndpoint to process and send the command as complete message
- When connection is made then the Connector sends the REG message to the Test Server to indicate it's presence

- The Colibri Test Server now sends the STA message to the Connector and the connector gets registered. Now, connector can send two messages(ADDT for adding the temperature service or ADDL for adding the light service)
- When the service is added, the Test Server can send REMT(REML), OBST(OBSL), DETT(DETL), GETT(GETL) to the connector to remove the service, observe the service, dettach the observation, get the service, etc
- When a service is being observed then for every change in the value of the service, the TEST server is notified
- In the similar way,other message transfer takes place according to the Colibri's semantic interface specification document

Chapter 4

Tools and Technologies used

4.1 API(Application Programming Interface)

4.1.1 javax.json API

This provides an object model API to process JSON. The object model API is a high-level API that provides immutable object models for JSON object and array structures. These JSON structures are represented as object models using the Java types JsonObject and JsonArray.

The interface javax.json.JsonObject provides a Map view to access the unordered collection of zero or more name/value pairs from the model. Similarly, the interface JsonArray provides a List view to access the ordered sequence of zero or more values from the model.

The object model API uses builder patterns to create these object models. The classes JsonObjectBuilder and JsonArrayBuilder provide methods to create models of type JsonObject and JsonArray respectively.

These object models can also be created from an input source using the class JsonReader. Similarly, these object models can be written to an output source using the class JsonWriter.

4.1.2 javax.websocket API

This package contains all the WebSocket APIs common to both the client and server side. Following annotations are used in all the WebSocket implementation:

- **ServerEndpoint:** This class level annotation declares that the class it decorates is a web socket endpoint that will be deployed and made available in the URI-space of a web socket server
- **ClientEndpoint:** The ClientEndpoint annotation a class level annotation is used to denote that a POJO is a web socket client and can be deployed as such
- **OnOpen:** This method level annotation can be used to decorate a Java method that wishes to be called when a new web socket session is open
- **OnClose:** This method level annotation can be used to decorate a Java method that wishes to be called when a web socket session is closing
- **OnMessage:**This method level annotation can be used to make a Java method receive incoming web socket messages
- **OnError:** This method level annotation can be used to decorate a Java method that wishes to be called in order to handle errors

4.2 Server

4.2.1 Glassfish application server

GlassFish is an open-source application server project started by Sun Microsystems for the Java EE platform and now sponsored by Oracle Corporation. The supported version is called Oracle GlassFish Server. GlassFish is free software, dual-licensed under two free software licences: the Common Development and Distribution License (CDDL) and the GNU General Public License (GPL) with the classpath exception.

4.2.2 Grizzly Framework

Grizzly is built, assembled and installed using Maven. Grizzly is deployed to the Maven Central repository. Binary, source, javadoc, and sample JARS can all be found there.

An application depending on Grizzly requires that it in turn includes the set of jars that Grizzly depends on. Grizzly has a pluggable component architecture so the set of jars required to be include in the class path can be different for each application.

All Grizzly components are built using Java SE 6 compiler. It means, you will also need at least Java SE 6 to be able to compile and run your application.

Developers using maven are likely to find it easier to include and manage dependencies of their applications than developers using ant or other build technologies. This document will explain to both maven and non-maven developers how to depend on Jersey for their application. Ant developers are likely to find the Ant Tasks for Maven very useful.

If you are not using maven, you can download the JARs you need for your project from maven central.

Chapter 5

Todo tasks

MATLAB Simulink connector is sophisticated and to implement it completely during Google Summer of Code period was not possible as Simulink can run any complex simulation and it can have 'n' number of services to offer to Colibri. Following are the tasks that needs to be done:

1. WebSocket security needs to be implemented
2. Connector authentication(only authenticated connector should be able to connect to Colibri) needs to be implemented
3. Only those messages that are in XML format are handled in Colibri. This should be upgraded to handle messages in JSON format and any other formats
4. The connector should be able to send SPARQL queries to Colibri semantic core to request data from the ontology. As of now, the queries are not communicating with the real time ontologies which also needs to be implemented