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 Fruehwirth

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.

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.

Chapter 2

Project components

The project makes use of MATLAB Simulink, 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, enabling 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.

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,

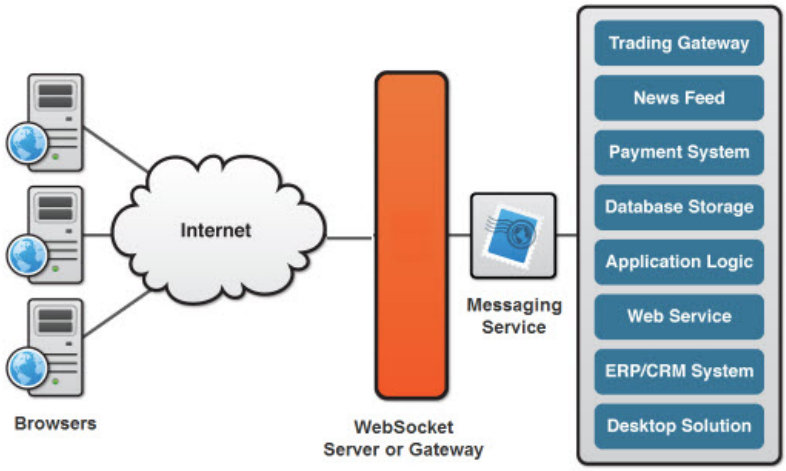

Figure 2.1: WebSocket Architecture

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

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.

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 Connector components and design

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

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

2.4.1 Colibri Demo Server

Colibri Demo Server (which will be replaced by actual Colibri Semantic Core) opens the WebSocket communication for its clients.Communication with the running Simulink is hidden by the connector and thus not specified by the Colibri semantic interface.

2.4.2 Connector

- Connector in the implementation is the ClientEndpoint for the Colibri Demo Server i.e. the Connector acts as a client to the Colibri Demo Server
- When connector is connected to the Colibri Demo Server then it opens the ServerSocket port for MATLAB to connect to it.
- In short, Connector acts as the WebSocket client to Colibri Demo 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 exchange of process data can take 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. This simulation is used for evaluation the implemented connector. However, the connector should not be limited to this simulation.

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, follow the steps in the mentioned order:

- Run the Project which will start the Colibri Demo Server using following commands in given order:
  1. Start the glassfish server
  2. gradle clean
  3. gradle build (This will create a task named ColibriJar)
  4. gradle ColibriJar (This will download all the dependencies to the local machine)
  5. pathtoGlassFishServer/bin ./asadmin deploy build/libs/colibri-simulink-1.0.war (asadmin is the shell script located in the bin folder of the Glassfish server and it is used to deploy the application war. In linux, to use asadmin, write ./asadmin deploy pathToWarFile to deploy the application)
  6. Go to http://localhost:8080/colibri-simulink-1.0/
- Run the Client.java file for Connector to connect as WebSocket client to the Colibri Demo Server and which opens the Socket port for MATLAB Simulink to connect to it. Do this using following command:

1. java -jar build/libs/colibri-simulink-all-1.0.jar

- Run the .m 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 Demo 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 Demo Server to indicate it's presence
- The Colibri Demo 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 Demo 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 value of the service
- When a service is being observed then for every change in the value of the service, the demo 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)

- **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
- **javax.websocket API**: This package contains all the WebSocket APIs common to both the client and server side like OnOpen, OnClose, OnMessage, OnError
- **org.glassfish.tyrus API**: Tyrus is built, assembled and installed using Maven. Jars, Jar sources, Jar JavaDoc and samples are all available on the Maven Central repository. An application depending on Tyrus requires that it in turn includes the set of jars that Tyrus depends on. Tyrus has a pluggable component architecture so the set of jars required to be include in the class path can be different for each application.
- **org.glassfish.grizzly API** This JAR is the minimum requirement for all Grizzly applications. This provides all core services: TCP/UDP transports, memory management services/buffers, NIO event loop/filter chains/filters.

4.2 Server

- **Glassfish application server**: GlassFish is an open-source application server project started by Sun Microsys- tems 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

- **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

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 most important tasks that needs to be done:

1. WebSocket security needs to be implemented(TLS/SSL)
2. Currently, the connector is configured for the simple demo simulation. However, an easy way of configuring the connector for different simulations needs to be implemented.
3. **Connector authentication**(only authenticated connector should be able to connect to Colibri) needs to be implemented
4. 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 format that is specified in the interface specification.
5. 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