Lecture – 08

Intro. to Internet of Things

Dr. Ahmed Elngar
Faculty of Computers and Artificial Intelligence
Beni-Suef University

19. IoT – Salesforce

The Salesforce IoT Cloud is a platform for storing and processing IoT data. It uses the Thunder engine for scalable, real-time event processing. Its collection of application development components, known as Lightning, powers its applications. It gathers data from devices, websites, applications, customers, and partners to trigger actions for real-time responses.

Salesforce, a CRM leader, decided to enter this space due to the need to remain competitive in the coming era. The IoT cloud adds to Salesforce by expanding its reach, and the depth of its analytics.

Salesforce combined with IoT delivers dramatically improved customer service with tighter integration and responses to real-time events; for example, adjustments in wind turbines could trigger automatic rebooking of delayed/canceled connecting flights before airline passengers land.

Electric Imp

The Electric Imp platform is Salesforce's recommended method for quickly connecting devices to the cloud. You develop applications through the Squirrel language; a high level, OO, lightweight scripting language. Applications consist of two modules: the device module, which runs on the device; and the agent module, which runs in the Electric Imp cloud. The platform ensures secure communication between the modules, and you send devices messages with a simple call:

```
agent.send("nameOfmessage", data);
```

Listen for messages on the agent with the following code:

```
device.on("nameOfmessage", function(data) {
  //Data operations
});
```

Beyond these basic tasks, coding for device interaction, monitoring, and response resembles standard web application development, and uses a simple, easy-to-learn syntax.

21. IoT – Eclipse IoT

Eclipse IoT is an ecosystem of entities (industry and academia) working together to create a foundation for IoT based exclusively on open source technologies. Their focus remains in the areas of producing open source implementations of IoT standard technology; creating open source frameworks and services for utilization in IoT solutions; and developing tools for IoT developers.

Smarthome Project

SmartHome is one of Eclipse IoT's major services. It aims to create a framework for building smart home solutions, and its focus remains heterogeneous environments, meaning assorted protocols and standards integration.

SmartHome provides uniform device and information access to facilitate interaction between devices. It consists of OSGi bundles capable of deployment in an OSGi runtime, with OSGi services defined as extension points.

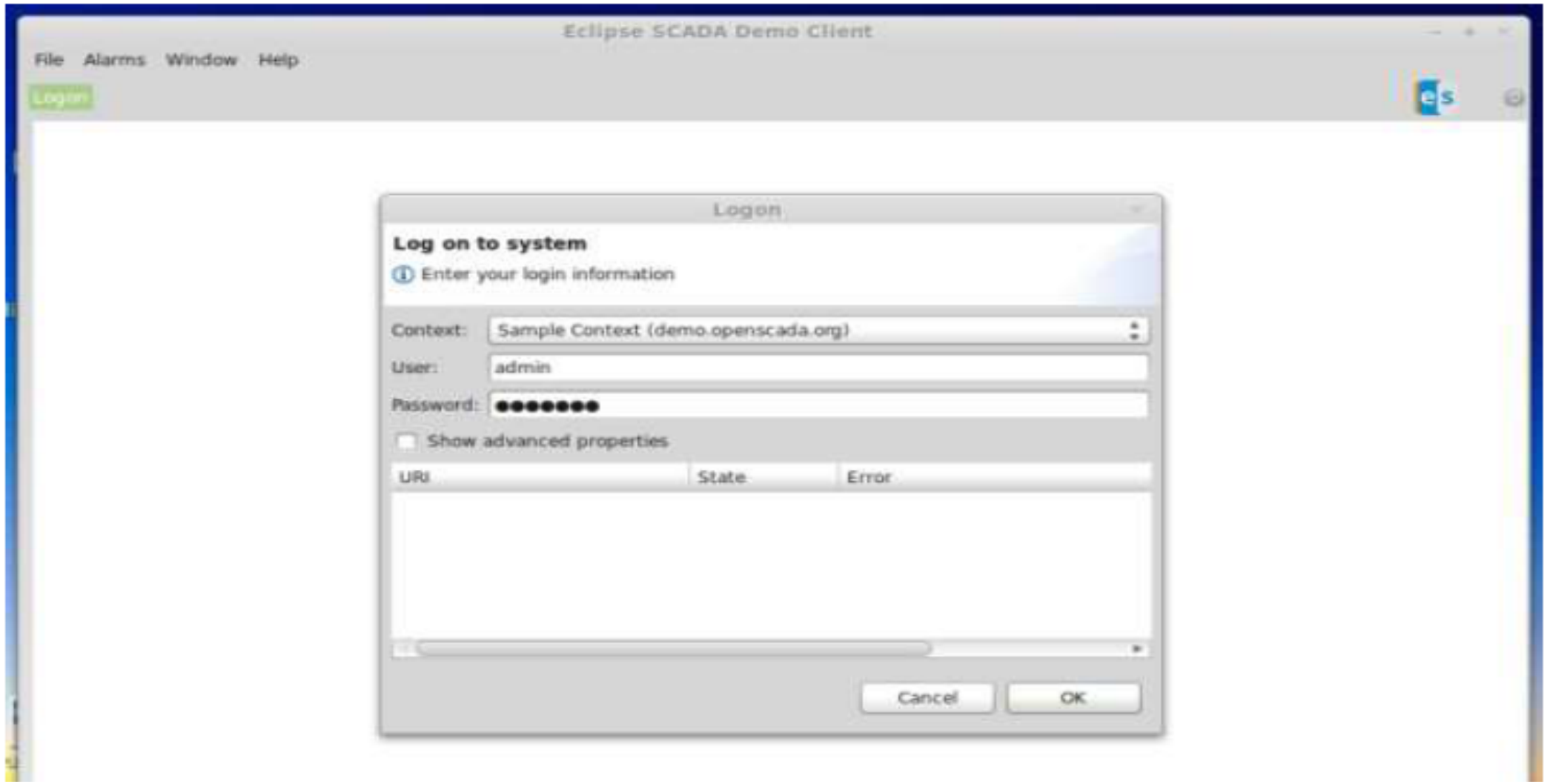
OSGi bundles are Java class groups and other resources, which also include detailed manifest files. The manifest contains information on file contents, services needed to enhance class behavior, and the nature of the aggregate as a component. Review an example of a manifest below:

```
Bundle-Name: Hi Everyone                        // Bundle Name
Bundle-SymbolicName: xyz.xyz.hievery1           // Header specifying an identifier
Bundle-Description: A Hi Everyone bundle        // Functionality description
Bundle-ManifestVersion: 2                       // OSGi specification
Bundle-Version: 1.0.0                           // Version number of bundle
```

```
Bundle-Activator: xyz.xyz.Activator                // Class invoked on bundle activation
Export-Package: xyz.xyz.helloworld;version="1.0.0"   // Java packages available externally
Import-Package: org.osgi.framework;version="1.3.0"   // Java packages needed from
                                                    // external source
```

Eclipse SCADA

Eclipse SCADA, another major Eclipse IoT service, delivers a means of connecting various industrial instruments to a shared communication system. It also post-processes data and sends data visualizations to operators. It uses a SCADA system with a communication service, monitoring system, archive, and data visualization.



It aims to be a complete, state-of-the-art open source SCADA system for developing custom solutions. Its supported technologies and tools include shell applications, JDBC, Modbus TCP and RTU, Simatic S7 PLC, OPC, and SNMP.

22. IoT – Contiki

Contiki is an operating system for IoT that specifically targets small IoT devices with limited memory, power, bandwidth, and processing power. It uses a minimalist design while still packing the common tools of modern operating systems. It provides functionality for management of programs, processes, resources, memory, and communication.

It owes its popularity to being very lightweight (by modern standards), mature, and flexible. Many academics, organization researchers, and professionals consider it a go-to OS. Contiki only requires a few kilobytes to run, and within a space of under 30KB, it fits its entire operating system: a web browser, web server, calculator, shell, telnet client and daemon, email client, vnc viewer, and ftp. It borrows from operating systems and development strategies from decades ago, which easily exploited equally small space.

Contiki Communication

Contiki supports standard protocols and recent enabling protocols for IoT:

- **uIP (for IPv4)** – This TCP/IP implementation supports 8-bit and 16-bit microcontrollers.
- **uIPv6 (for IPv6)** – This is a fully compliant IPv6 extension to uIP.
- **Rime** – This alternative stack provides a solution when IPv4 or IPv6 prove prohibitive. It offers a set of primitives for low-power systems.
- **6LoWPAN** – This stands for IPv6 over low-power wireless personal area networks. It provides compression technology to support the low data rate wireless needed by devices with limited resources.
- **RPL** – This distance vector IPv6 protocol for LLNs (low-power and lossy networks) allows the best possible path to be found in a complex network of devices with varied capability.
- **CoAP** – This protocol supports communication for simple devices, typically devices requiring heavy remote supervision.

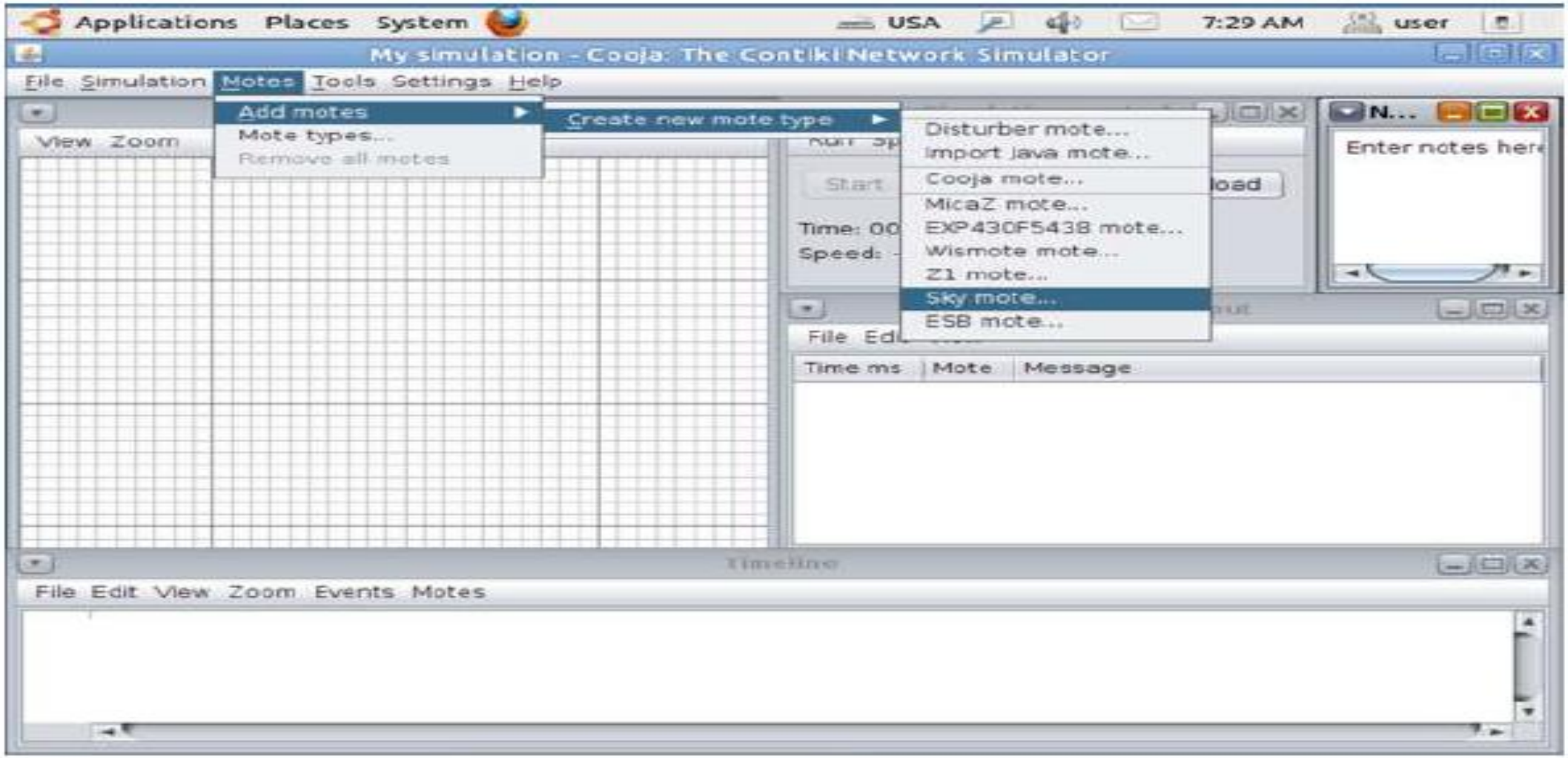
Dynamic Module Loading

Dynamic module loading and linking at run-time supports environments in which application behavior changes after deployment. Contiki's module loader loads, relocates, and links ELF files.

The Cooja Network Simulator

Cooja, the Contiki network simulator, spawns an actual compiled and working Contiki system controlled by Cooja.

Using Cooja proves simple. Simply create a new mote type by selecting the **Motes** menu and **Add Motes > Create New Mote Type**. In the dialog that appears, you choose a name for the mote, select its firmware, and test its compilation.



After creation, add motes by clicking **Create**. A new mote type will appear to which you can attach nodes. The final step requires saving your simulation file for future use.