JavaOne™

ORACLE

Pushing Java EE outside of the Enterprise

Home Automation

David Delabassee - @delabassee
Software Evangelist – Java EE
Oracle
September, 2014

JavaOne ORACLE

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Program Agenda

1. Home Automation
2. Java EE
3. Demo
4. Wrap-up

Program Agenda

1. Home Automation
2. Java EE
3. Demo
4. Wrap-up

Home Automation

Features

- “Devices” control
  - Lights, windows, blinders, audio, doors, wall-plugs, heating, etc.
  - From different “terminal”
- Monitoring & dashboard
  - Power consumption, weather, motion/presence, intrusion, etc.
- Scenario, scene, scheduling & automation
- “Flexible” configuration

Home Automation

Benefits

- Better control
  - Improves day-to-day life, e.g.. disabled person
  - Improves security
  - More Eco friendly
- …

Home Automation

Market

- Crowded space
- Just the beginning!
- IoT

Home Automation

Technologies

- X10 / PCLBUS
- RF433
- EnOcean
- Z-Wave
- ZigBee
- Domologic
- Domintell
- NikoBus / HomeControl
- Chacon / DI-O
- Somfy RTS
- io-homecontrol
- VelBus
- Domologic
- KNX
- BLE
- HomeKit
- Thread
- …

Home Automation

Actors

- Sensors
  - Switches, motion, temperature, CO2, humidity, wind speed, etc.
- Actuators
  - Lights, blinders, windows, doors, audio, wall-plugs, etc.
- Network
  - Wired or/and wireless + TCP/IP
- Hub, box, gateway
  - Provides added values

Z-Wave

“Z-Wave, The Interoperable Standard”

- “Interoperable wireless RF-based communications technology designed for control, monitoring and status reading applications in residential and light commercial environments.”
- Proprietary
- Z-Wave Alliance
  - Consortium of 250 manufacturers and service providers worldwide

http://www.z-wavealliance.org

Z-Wave

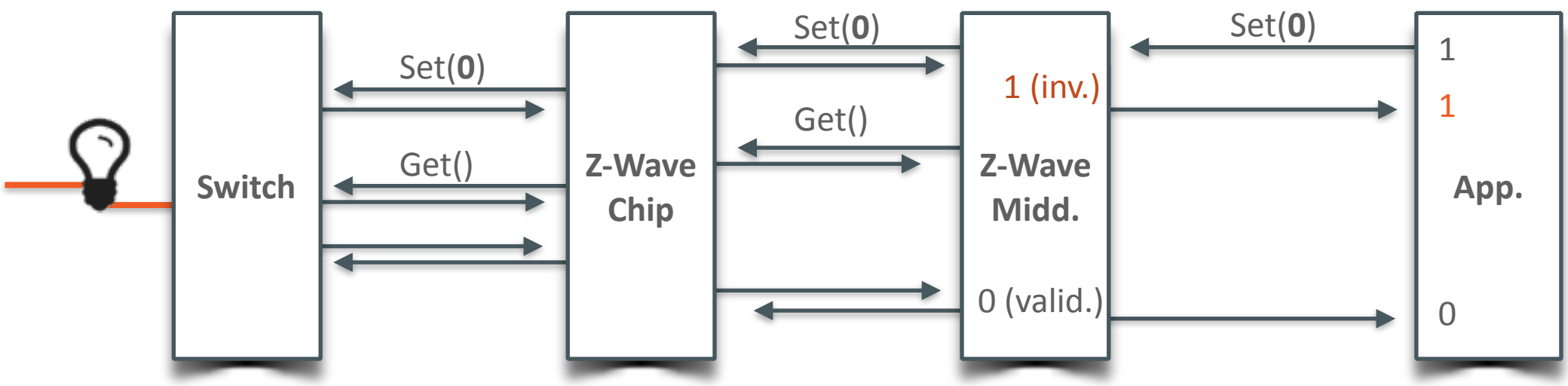
“Z-Wave, The Interoperable Standard”

- Meshed wireless network
- Range 100 ft (open-sight)
- 232 modules per controller
- Cheap and simple

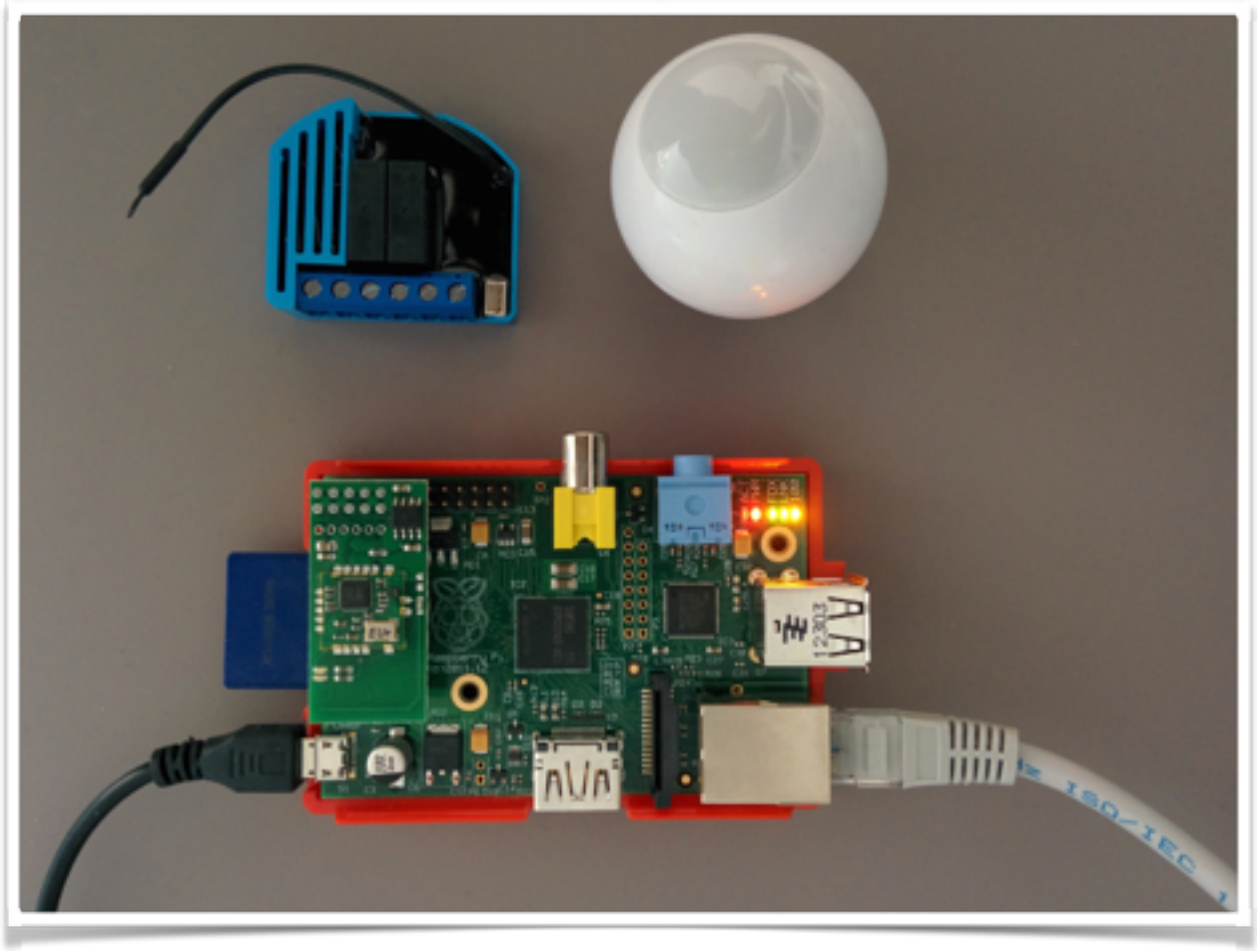
Connectivity

- Physical connectivity
  - Razberry-Pi Daughter Card
  - Aeon Labs Z-Stick, …
- Middleware
  - OpenZwave
  - RaZberry / Z-Wave-me / Z-Way
  - Domoticz, …

Principles



In real life



KNX

“The worldwide STANDARD for home and building control”

- ISO/IEC 14543-3
- KNX Association
  - 340 KNX Members in 37 countries
  - 40 KNX National Groups
  - 275 Training Centres in 51 countries
  - 40,189 KNX Partners in 125 countries
  - 100 Scientific Partners in 28 countries
  - 13 User clubs in 12 countries

http://www.knx.org

KNX

“The worldwide STANDARD for home and building control”

- Media
  - TP, PL, RF and IP
- Engineering Tool Software
  - Manufacturer independent configuration software tool
  - Design and configure intelligent home and building KNX installations

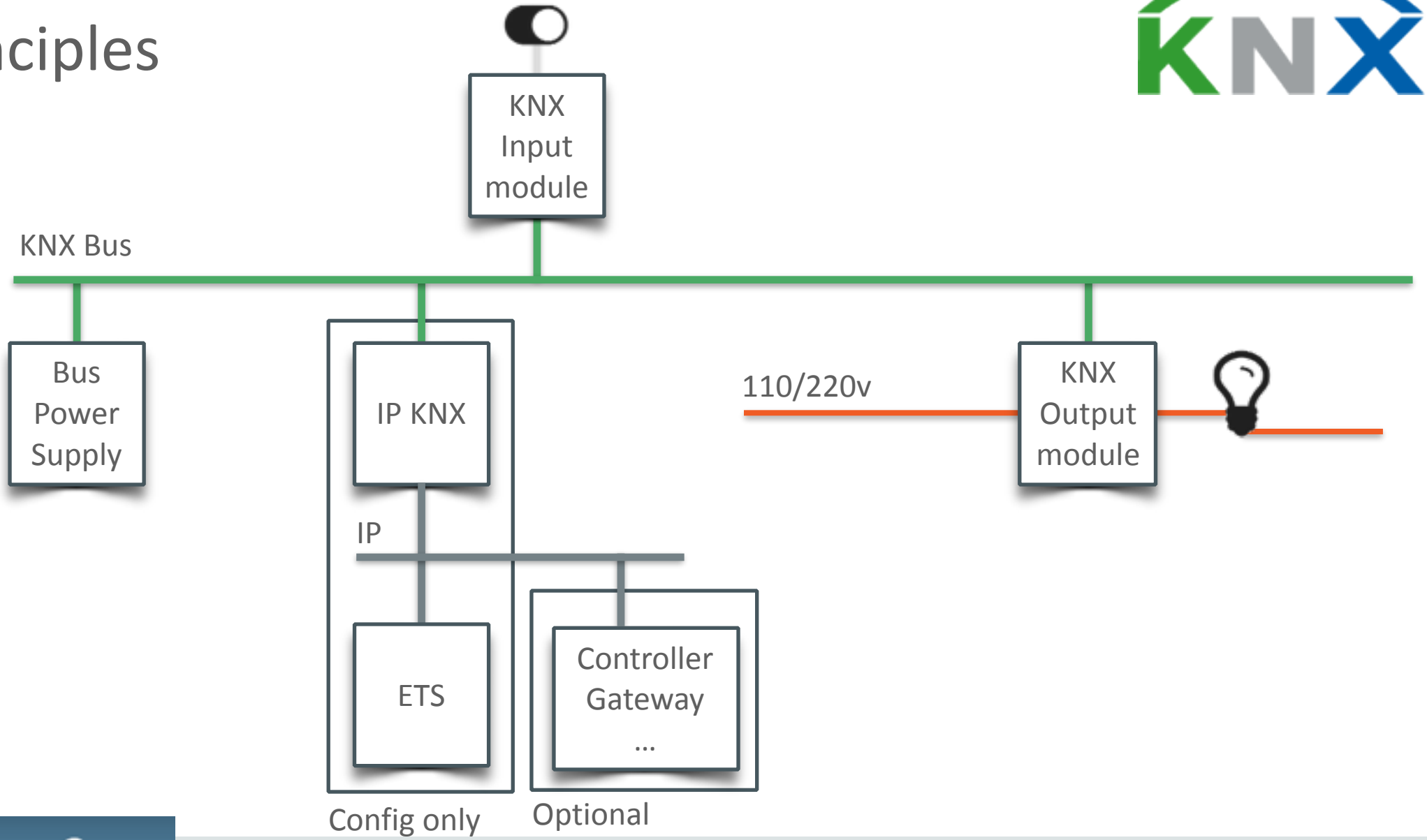
Connectivity

- Physical connectivity
  - KNX/USB Router
  - KNX/IP Router
  - Eelectron Raspberry Pi KNX interface

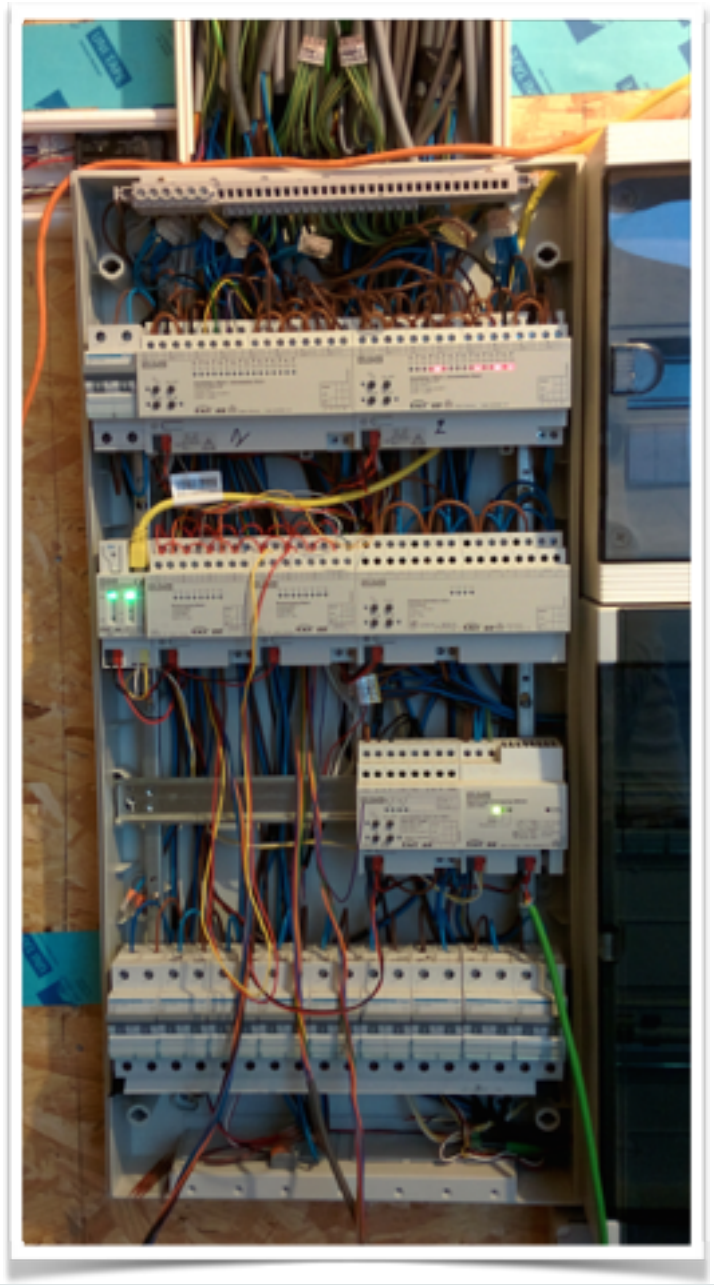
Calimero

- Java library for KNX/EIB applications
  - KNX Network services and data encodings
  - Routing
  - Local device management
  - Datapoint type and property type translation
  - ETS4 import, …
- Min Java 2 Micro-Edition CDC Foundation Profile
- http://calimero.sourceforge.net

Principles



In real life



Z-Wave Vs. KNX

- Proprietary
- 232 devices per controller
- Residential grade
- Wireless, more sensible
- Powered via batteries and PL
- Controller is the Single PoF
- Simple, non intrusive
- Cheap

- ISO Standard
- Up to 58.000 bus devices
- Industrial, professional grade
- Wired, more reliable
- Powered through the Bus
- No real PoF
- Complex, intrusive
- Expensive

Program Agenda

1. Home Automation
2. Java EE
3. Demo
4. Wrap-up

Java EE

- More annotated POJOs
- Less boilerplate code
- Cohesive integrated platform

- WebSockets
- JSON
- Servlet 3.1 NIO
- REST

- Batch
- Concurrency
- Simplified JMS

Java EE

- Connectivity
  - EE to clients
  - EE to HA
- Events

JSON-P

JSR 353

- API to parse and generate JSON
- Streaming API (`javax.json.stream`)
  - Low-level, efficient way to parse/generate JSON
  - Similar to StAX API in XML world
- Object model API (`javax.json`)
  - Simple, easy to use high-level API
  - Similar to DOM API in XML world

JSON-P

```
public class Notification {

    // bunch of Getters & Setters

    public String toJson() {
        JsonObject model = Json.createObjectBuilder()
                .add("source", getSource())
                .add("target", getTarget())
                .add("val", getValue())
                .add("type", getType())
                .add("medium", getMedium())
                .add("cyclic", isCyclic())
                .add("time", getTimestamp())
                .build();
        return model;
}
```

Java API for Web Socket

JSR 356

- Bidirectional full-duplex messaging
  - Initial HTTP handshake
  - Over a single TCP connection
- IETF defined protocol: RFC 6455
- HTML5 / W3C defined JavaScript API
- JSR 356
  - Client & Server API

WebSocket Server Endpoint (1/2)

```
@ServerEndpoint("/wsEndPt")
public class InboundWebsocket implements Serializable {

    static Queue<Session> queue = new ConcurrentLinkedQueue<>();

    @OnOpen
    public void openConnection(Session session) {
        queue.add(session);
    }

    public void onEvent(Notification notif) {
        sendAllEndpoints(notif.toJson());
    }

    …
```

WebSocket Server Endpoint (2/2)

```
…
public static synchronized void sendAllEndpoints(String payload) {
     try {
         for (Session session : queue) {
             if (session.isOpen()) {
                 session.getBasicRemote().sendText(payload);
             }
         }
     } catch (IOException e) {
         // Oooops
     }
 }
```

WebSocket Client Endpoint

```
function connect() {
    wsocket = new WebSocket('ws://' + window.location.host + '/dashboard/wsEndPt');
    wsocket.onmessage = onMessage;
}

function onMessage(evt) {
   jsonData = JSON.parse(evt.data);
   targetDiv = jsonData.target;
   var myDiv = document.getElementById(targetDiv);

   if (jsonData.type === "switch") {
          if (jsonData.val === "true") value = "ON";
          else value = "OFF";
   }
   else value = jsonData.val;
   $(myDiv).text(value);
}
```

Contexts and Dependency Injection for Java EE 1.1

JSR 346

- Core component model
  - Enabled by default in implicit bean archives
- Expanded use of CDI Interceptors
  - Transactional interceptors
  - Method-level validation interceptors
- New CDI scopes
  - @TransactionScoped
  - @FlowScoped

CDI Events

JSR 346

- Observer Design Pattern
- Defined by
  - A java object (the event object)
  - Event Qualifier(s)

CDI Events

JSR 346

```
@Inject
BeanManager beanManager;

public void doSomething() {
      beanManager.fireEvent(new Payload());
}



public void listenToPayload(@Observes Payload event) {
      //do something using the payload
}
```

CDI Events

MDB

```
@MessageDriven(
    …
)
public class KnxMdb implements KnxListener {

    @Inject
    @HomeEvent
    Event<Notification> homeEvent;

    @KnxMessage(type="sensor")
    public void handleKnxSensor(Notification notif) {
        …
        homeEvent.fire(notif);
    }


}
```

CDI Events

WebSocker Server Endpoint

```
@ServerEndpoint("/wsEndPt")
public class InboundWebsocket implements Serializable {


    @Inject
    @HomeEvent
    Event<Notification> homeEvent;

    @OnOpen
    public void openConnection(Session session) {
        …
    }

    public void onEvent(@Observes @HomeEvent Notification notif) {
        sendAllEndpoints(notif.toJson());
    }
    …
```

Java API for RESTful Web Services 2.0

JSR 339

- Filters and Interceptors
- Asynchronous Processing
- Hypermedia
- Validation
- Client API

JAX-RS Client

```
private void zwaveCommand(String url, String node, boolean val) {

    //http://host/ZWaveAPI/Run/devices[3].instances[0].commandClasses[0x25].Set(false)

    String zwaveUrl = url + node + ".Set(" + val + ")";

    private final Client jaxrsClient = ClientBuilder.newClient();

    String zwaveResp = jaxrsClient.target(zwaveUrl)
            .request(MediaType.TEXT_PLAIN_TYPE)
            .get(String.class);
}
```

REST Endpoint

```
@Path("/zwave")
public class ZwaveEndpoint {

    @Inject
    @HomeEvent
    Event<Notification> homeEvent;

    @PUT
    @Path("/{id}/{val}/")
    @Produces("text/plain")
    public String swithKnx(@PathParam("id") String id, @PathParam("val") String val) {
        Notification notif = new Notification(id, "sensor" , val, "ZWAVE");
        notif.setTarget(id); // target is required to identify the Zwave device
        notif.setValue(val);
        homeEvent.fire(notif);
    }

}
```

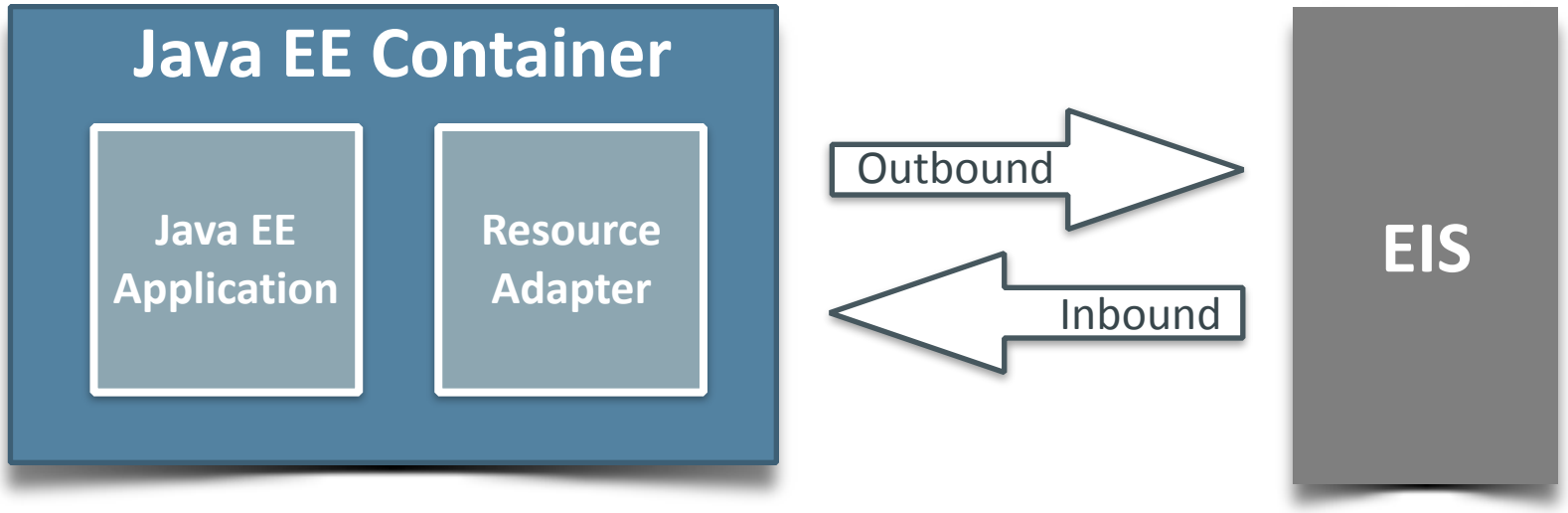
Java Connector Architecture 1.7

JSR 322

- Resource Adapter
  - JCA component for a specific EIS
  - Mediates communication between the container and the **EIS** by means of **contracts**
  - Packaged as a RAR
- EIS
  - Packaged Applications, ERP, CICS, mail server, MoM, etc.
- Contracts
  - Lifecycle, Connection (inc. pooling), TX, Work Management, CCI, …

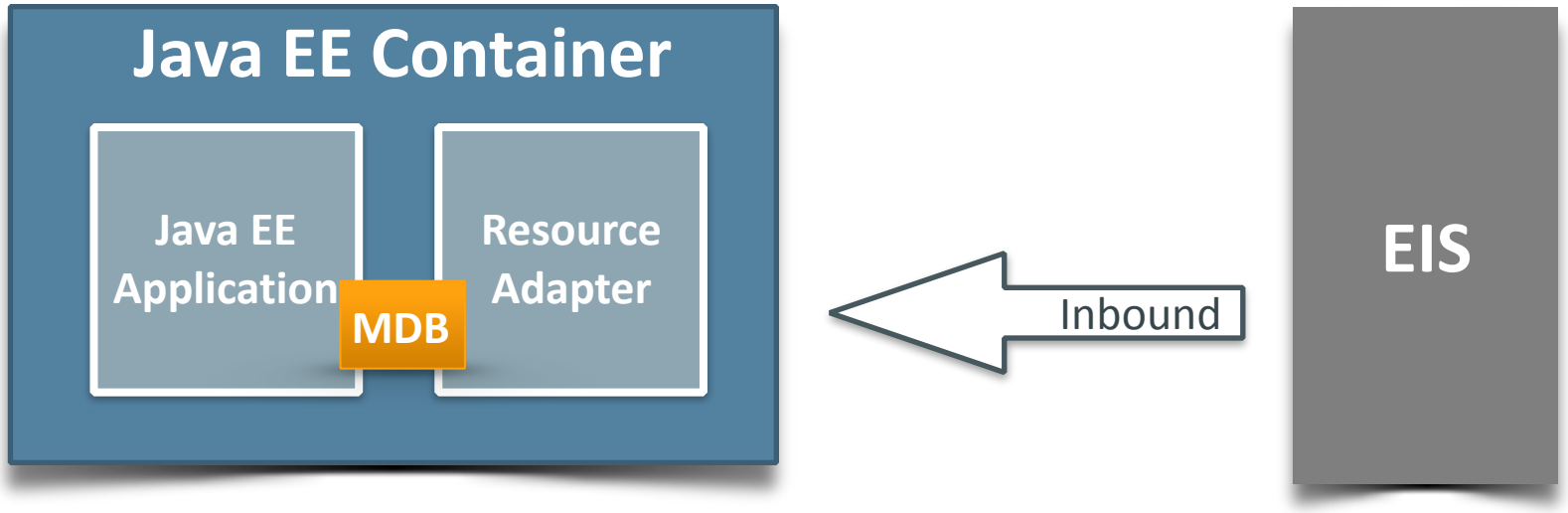
Java Connector Architecture 1.7

JSR 322



Java Connector Architecture 1.7

JSR 322



Message Driven Bean

JMS

```
@MessageDriven (
     activationConfig = {
          @ActivationConfigProperty(
               propertyName="destinationLookup", propertyValue="…"),
          @ActivationConfigProperty(
               propertyName="destinationType", propertyValue="javax.jms.Queue")
        }
)

public class MyJmsMdb implements MessageListener {

     @Override
     public void onMessage(Message message) {
     //do the work
     }

}
```

Message Driven Bean

- *“A MDB is an asynchronous message consumer…”*
- *“A MDB is invoked by the container as a result of the arrival of a message at the destination or endpoint that is serviced by the MDB…”*
- *“MDB listen to an Inbound Ressource Adapter”*
  - Eg. JMS

Message Driven Bean

KNX

```
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
              propertyName = “knx", propertyValue = "10.0.0.60")
    }
)
public class KnxMdb implements KnxListener {

     @Override
     public void onMessage(Message message) {
     //handle the KNX Message
     }

}
```

Message Driven Bean

KNX

```
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
                propertyName = “knx", propertyValue = "10.0.0.60")
    }
)

public class KnxMdb implements KnxListener {

    @KnxMessage(type="sensor")
    public void handleKnxSensor(Notification notif) {
        …
    }

    @KnxMessage(type=“switch")
    public void switchKnx(Notification notif) {
        …
    }

}
```

Inbound RA

- `public interface KnxListener { }`
  - Will be implemented by the MDB
- `public class KnxResourceAdapter implements ResourceAdapter {…}`
  - Do the actual work, without real constraints (e.g. threads)
  - Handle MDB activation
  - Will listen on the KNX bus via WorkManager
- `public class KnxActivationSpec implements ActivationSpec {…}`
  - Expose RA configuration to the application

Inbound RA

```
@Override
public void endpointActivation(MessageEndpointFactory endpointFactory,
                               ActivationSpec spec)
                               throws ResourceException {

    tSpec = (TrafficActivationSpec) spec;
    Class endpointClass = endpointFactory.getEndpointClass();
    tSpec.setBeanClass(endpointClass);
    tSpec.findCommandsInMDB();

    ObtainEndpointWork work = new ObtainEndpointWork(this, endpointFactory);
    workManager.scheduleWork(work);
}
```

Inbound RA

```
public void findCommandsInMDB() {

    for (Method method : beanClass.getMethods()) {
        if (method.isAnnotationPresent(KnxMessage.class)) {
            KnxMessage tCommand = method.getAnnotation(KnxMessage.class);
            commands.put(tCommand.type(), method);
        }
    }

   …
}
```

KNX (real) Listener

```
// KNX ProcessedEvent from the Calimero stack
switch (actor) {
            case "0/7/4":
                messageType = "sensor";
                notif.setType("humidity");
                processedEvent = Float.toString(knxSensorHumidity(pe));
                notif.setValue(processedEvent); // and so on…
                break;
}

if (spec.getCommands().containsKey(messageType)) { // Does the MDB support this message?
     Method mdbMethod = spec.getCommands().get(messageType);
     try {
          callMdb(mdb, mdbMethod, notif);
     } catch (ResourceException ex) { // Ooops! }
} else { // Unknown command ?! }
```

KnxMessage

```
/* Annotation to decorate methods in the MDB */
@Target({ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface KnxMessage {
          String type() default "";
          String info() default "";
}
```

Inbound connector

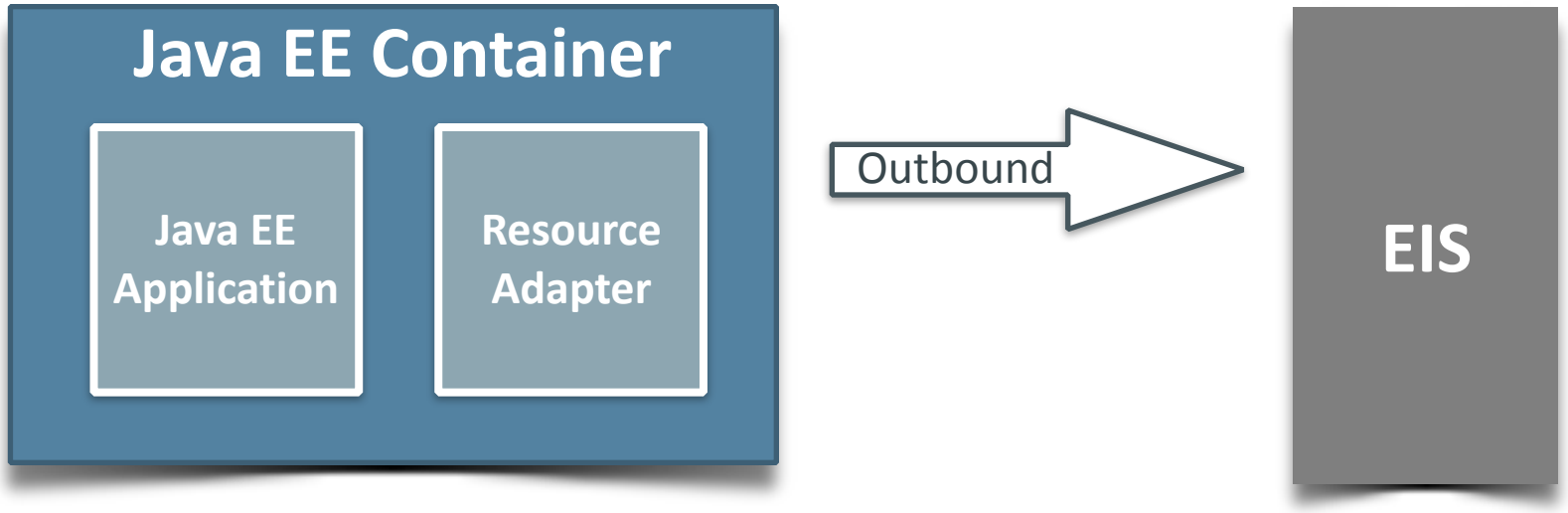
```
private Notif callMdb(MessageEndpoint mdb, Method command, Notification param)
            throws ResourceException {

     try {
           mdb.beforeDelivery(command);
           Object ret = command.invoke(mdb, param);
           notif = (Notif) ret;
     } catch (NoSuchMethodException | ResourceException | IllegalAccessException |
              IllegalArgumentException |InvocationTargetException ex) {
           // MDB invocation error :(
           …
     }
     mdb.afterDelivery();
     return notif;
}
```

KNX MDB

```
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
              propertyName = “knx", propertyValue = "10.0.0.60")
    }
)

public class KnxMdb implements KnxListener {

    @KnxMessage(type="sensor")
    public void handleKnxSensor(Notification notif) {
        …
    }

    @KnxMessage(type=“switch")
    public void switchKnx(Notification notif) {
        …
    }

}
```

Java Connector Architecture 1.7

JSR 322



And more…

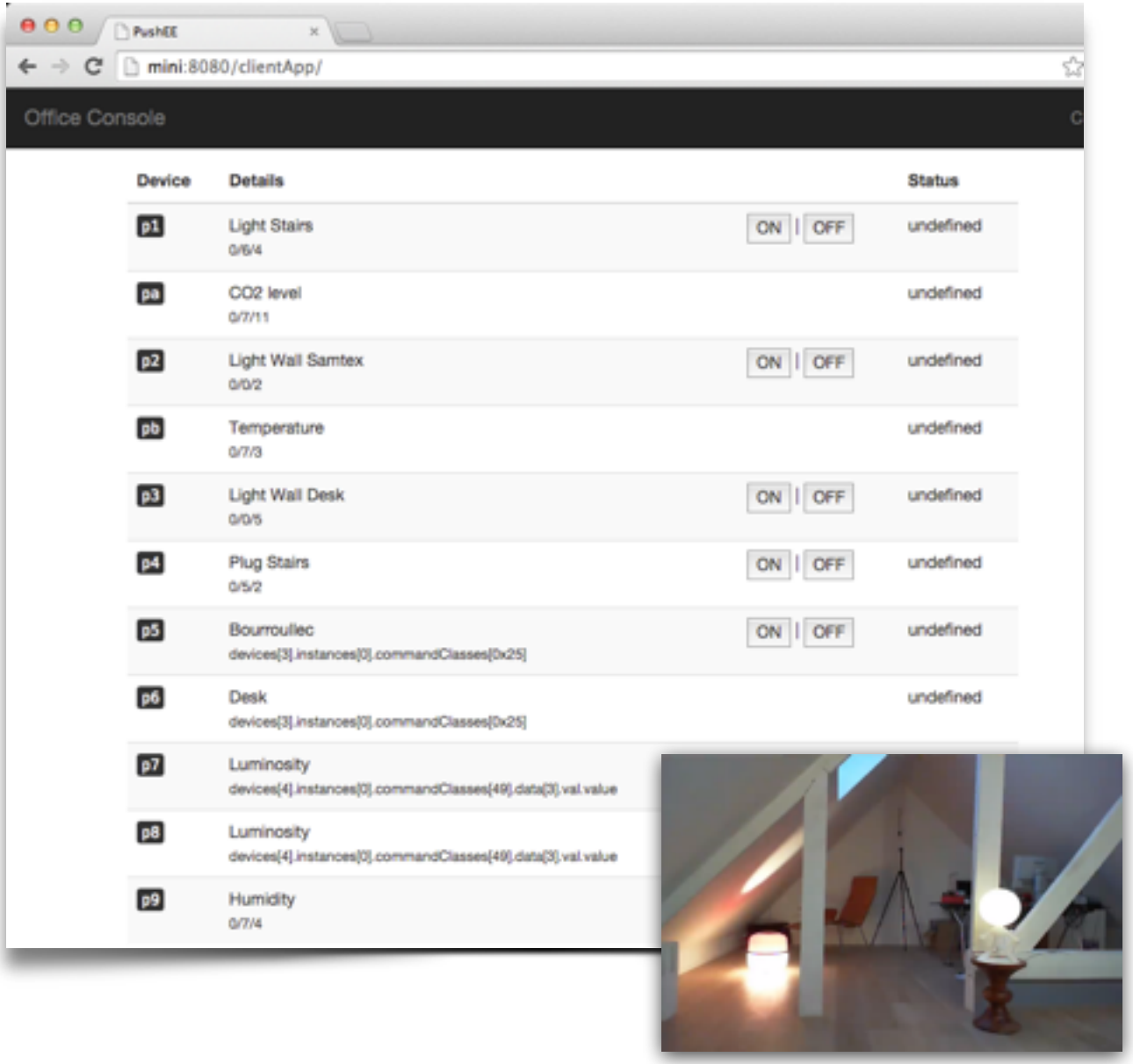
- EJB
- CDI Interceptors
- JAX-RS Interceptors
- Concurrency Utilities for Java EE
- Java Persistence API
- JavaServer Faces
- OpenMQ Stomp support
- Jersey SSE, MVC…

Program Agenda

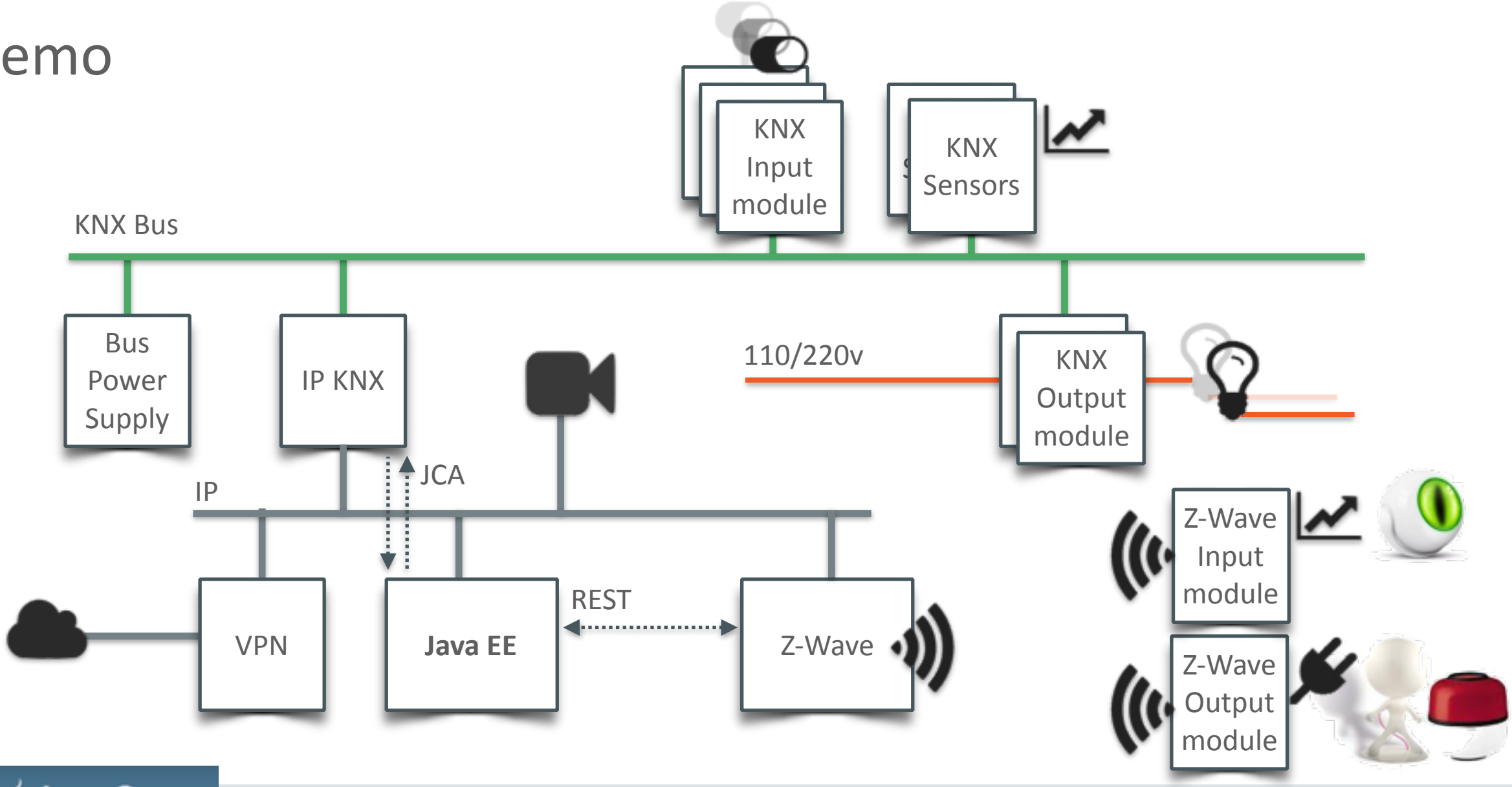
1. Home Automation
2. Java EE
3. **Demo**
4. Wrap-up

Dashboard

- HTML Client
  - “Pure” JS
  - Real-time notifications via WebSocket
  - Control devices via REST
- Java EE 7 backend
  - KNX (JCA) & Z-Wave (REST)
  - In & Outbound



Demo



Program Agenda

1. Home Automation
2. Java EE
3. Demo
4. Wrap-up

Java EE for Home Automation

- Connectivity
  - JAX-RS, JCA, WebSocket, JMS…
- ‘Event’ driven capabilities
  - MDB, CDI event
- Non blocking IO
- Asynchronous
- Scheduling
- Persistence

- Component models
- UI
  - JSF, JAX-RS
- Container
  - Lifecycle
  - Monitoring & Management
  - Ressources pooling
  - Logging
  - …

What's next

- PoC
  - KNX, Z-Wave & InfraRed
- Improvements
  - Additional 'connectors', e.g. RF433
  - More types, e.g. range for blinders
  - State Management
  - User interface
  - Add 'scenario's
  - Leverage Java EE more, e.g. CDI Interceptors

Java EE and IoT

- Connected devices
- Messages / data producer
- Data aggregation / processing
  - Deeper in the network
- Scaling
- Security

Java EE 8

TBC!

- CDI 2.0
- JMS 2.1
- MVC 1.0
- JAX-RS 2.1
- JSON-B + JSON-P

Resources

- The Java EE 7 Tutorial
  - http://docs.oracle.com/javaee/7/tutorial/doc/home.htm
- GlassFish 4.1
  - http://glassfish.org
- NetBeans 8.0.1
  - http://netbeans.org

JavaOne™

ORACLE


ORACLE®