EUSKO JAURLARITZA
GOBIERNO VASCO

Programa Saiotek 2006

SMARTLAB

Entorno de Trabajo Inteligente Colaborativo y Programable

Manual del programador de la plataforma semántica

Tecnológico Fundación Deusto
Teknologikoa Deustu Fundazioa

HISTORIAL DE CAMBIOS

Versión	Descripción	Autor	Fecha	Comentarios
V0.1	Versión inicial	Aitor Almeida	14/01/2008	

TABLA DE CONTENIDOS

1 INTRODUCCIÓN

En este documento se explica como desarrollar un nuevo bundle para la plataforma Smartlab. Los bundles deberán de ser capaces de semantizar su información, es decir, ampliar la ontología con clases que representen al dispositivo, sus eventos y su información, crear instancias de esas clases y proporcionar las reglas que puedan razonar con esos datos.

2 DESARROLLO DE UN BUNDLE SEMÁNTICO

2.1 Interfaz ISmartlabService

Todos los bundles deberán de implementar la interfaz ISmartlabService. Esta interfaz contiene los métodos que invocará el ContextManager para recuperar toda la información necesaria. La estructura de la interfaz es la siguiente:

```
public interface ISmartlabService {
	public String getOntology();
	public String getIndividual();
	public String getRules();
	public String[] getEventsToRegister();
	public void startUpdatingContext();
	public void stopUpdatingContext();
	public String getInterface();
}
```

- getOntology: devolverá una ontología con las clases del dispositivo, los valores y el evento en RDF/XML.

- getIndividual: devuelve una instancia de la clase del dispositivo con los valores para ese dispositivo en RDF/XML.

- getRules: devuelve las reglas específicas del dispositivo en el formato de reglas de JENA.

- getEventsToRegister: Devuelve los nombres de los eventos en los que está interesado el bundle.

- startUpdatingContext: el ContextManager llama a este método para que el bundle comience a actualizar sus medidas. Para enviar las medidas el bundle deberá llamar al método UpdateContext del interfaz IContextManager.

- stopUpdatingContext: el ContextManager llama a este método cuando el bundle debe de parar de actualizar el contexto.

- getInterface: devuelve el nombre de la interfaz del bundle.

A continuación se muestra un ejemplo del formato de la información enviada por un bundle, en este caso el bundle representa a un termómetro. Mediante la función getOntology el bundle devolvera las clases que representan a la temperatura y el evento de temperatura, no es necesaria una clase termómetro porque ya existe en la ontología base:

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns="http://www.morelab.deusto.es/smartlab.owl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://www.morelab.deusto.es/smartlab.owl">

  <!--TEMPERATURE ONTOLOGY-->
  <owl:Class rdf:ID="Temperature">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="temperatureValue"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="#ValueItem"/>
  </owl:Class>
  <owl:DatatypeProperty rdf:about="#temperatureValue">
    <rdfs:domain rdf:resource="#Temperature"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
  </owl:DatatypeProperty>

<!--EVENT ONTOLOGY -->
  <owl:Class rdf:ID="LocableTemperatureEvent">
    <rdfs:subClassOf rdf:resource="#LocableContextEvent"/>
  </owl:Class>
  <owl:FunctionalProperty rdf:ID="temperatureValue">
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float"/>
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    <rdfs:domain rdf:resource="#LocableTemperatureEvent"/>
  </owl:FunctionalProperty>
</rdf:RDF>
```

Cuando el ContextManager solicite el individuo del termómetro el bundle devolverá una instancia de la clase termómetro:

```
<!--THERMOMETER INDIVIDUAL-->
  <SpatialPoint rdf:ID="InsertPointIndividualName">
    <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >insertPointName</name>
    <x rdf:datatype="http://www.w3.org/2001/XMLSchema#float">55555.0</x>
    <y rdf:datatype="http://www.w3.org/2001/XMLSchema#float">666666.0</y>
    <containsDeviceItem>
      <Thermoter rdf:ID="InsertThermometerIndividualName">
        <name rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >InsertName</name>
        <id xml:lang="es">insertID</id>
        <deviceType rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
        >sensor</deviceType>
        <placedIn rdf:resource="#InsertPointIndividualName"/>
      </Thermoter>
    </containsDeviceItem>
  </SpatialPoint>
```

Las reglas estarán definidas en el formato de JENA, en esta caso la regla lanza un evento en caso de que la temperatura sobrepase un valor:

```
[EVENT-TemperatureHigh:
      (?temperature rdf:type <http://www.morelab.deusto.es/smartlab.owl#Temperature>),
      (?temperature <http://www.morelab.deusto.es/smartlab.owl#temperatureValue> ?value),
      (?temperature <http://www.morelab.deusto.es/smartlab.owl#targets> ?target),
      (?temperature <http://www.morelab.deusto.es/smartlab.owl#time> ?time),
      (?target rdf:type <http://www.morelab.deusto.es/smartlab.owl#DeviceItem>),
      (?target <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?place),
      greaterThan(?value, 30.0),
      makeTemp(?tempEvent)
      ->
      (?tempEvent rdf:type
      <http://www.morelab.deusto.es/smartlab.owl#LocableTemperatureEvent>),
      (?tempEvent <http://www.morelab.deusto.es/smartlab.owl#subject> ?target),
      (?tempEvent <http://www.morelab.deusto.es/smartlab.owl#time> ?time),
      (?tempEvent <http://www.morelab.deusto.es/smartlab.owl#place> ?place),
      (?tempEvent <http://www.morelab.deusto.es/smartlab.owl#temperatureValue> ?value)
      ]
```

El termómetro esta interesado en el evento producido por la regla, por lo tanto la función getEventsToRegister devolverá LocableTemperatureEvent. Además cuando comience a actualizar sus medidas el termómetro enviará individuos de la clase temperatura:

```
<!--TEMPERATURE INDIVIDUAL-->
  <Temperature rdf:ID="InsertTemperatureIndividualName">
    <targets rdf:resource="#InsertDeviceIndividualName"/>
```

```
    <temperatureValue rdf:datatype="http://www.w3.org/2001/XMLSchema#float"
    >7777777.0</temperatureValue>
    <time>
      <Instant rdf:ID="InsertTemperatureInstantName">
        <value rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
        >1950-11-11T11:11:11</value>
      </Instant>
    </time>
  </Temperature>
```

2.2 Activator

En su activator el bundle deberá realizar 3 acciones:

1 Registrar el interfaz ISmartlabService.

2 Registrar el interfaz del dispositivo (en el caso del termómetro IThermometer).

3 Recuperar el servicio IContextManager. Este servicio lo utilizará la implementación del dispositivo para actualizar la información de contexto.

2.3 Eventos

En el constructor de la implementación del dispositivo éste deberá de registrarse a los eventos que le interesa recibir. Para registrarse a eventos se utilizará el servicio de eventos de OSGi. Para ello el bundle deberá de implementar el interfaz EventHandler, de esta manera podrá recibir los eventos. Para registrarse a los eventos :

```
Hashtable ht = new Hashtable();
ht.put(EventConstants.EVENT_TOPIC, topics);
bundleContext.registerService(EventHandler.class.getName(), this, ht);
```

Para tratar los eventos:

```
public void handleEvent(Event arg0) {
      Vector<String> message = (Vector<String>)arg0.getProperty("ids");
      System.out.println("New Event received in Display: " + message.size());
}
```