EUSKO JAURLARITZA
GOBIERNO VASCO

Programa Saiotek 2006

SMARTLAB

Entorno de Trabajo Inteligente Colaborativo y Programable

Smartlab: Modelado semántico y reglas del sistema

Tecnológico Fundación Deusto
Teknologikoa Deustu Fundazioa

1. TECNOLOGÍAS UTILIZADAS

A la hora de desarrollar el sistema se ha hecho uso de diferentes tecnologías. Para poder diseñar la ontología se ha utilizado Protégé[PROT] al que se le ha añadido Pellet [PELL] como razonador externo para poder validar la misma. Para poder manipular desde código la ontología se ha hecho uso del Framework semántico Jena[JENA] y las consultas a la misma se han hecho mediante SPARQL[SPARQ].

JENA

Jena es un framework Java Open Source y gratuito que permite desarrollar aplicaciones Semánticas. Proporciona un API que permite manipular fácilmente RDF[RDF], OWL[OWL] y SPARQL. Además proporciona un motor de inferencia basado en reglas que permite el encadenamiento hacia delante, hacia atrás e híbrido. Jena incluye:

- Un API RDF.
- Funciones para exportar RDF a diferentes formatos: RDF/XML, N3, N-triples…
- Un API OWL.
- Almacenamiento persistente de las ontologías tanto en ficheros como en bases de datos.
- Un motor de reglas.
- Un motor de consultas para SPARQL.

Fig. 1 El framework semántico Jena

PROTÉGÉ

Protégé es un editor de ontologías y un framework para trabajar con bases de conocimiento Open Source y gratuito. Proporciona las herramientas necesarias para construir modelos de dominio y bases de conocimiento haciendo uso de ontologías a la vez que facilita su manipulación y visualización. Protégé permite modelar ontologías de dos formas diferentes:

- *Protégé-Frames:* Permite modelar ontologías de acuerdo al Open Knowledge Base Connectivity protocol [OKBC]. En este modelo uno ontología consiste en una jerarquía de clases que representan los conceptos más importantes del dominio.
- *Protégé-OWL:* Permite construir ontologías siguiendo el Web Ontology Lenguaje [OWL]. Según la OWL Web Ontology Language Guide[OWLGUI]: “An OWL ontology may include

descriptions of classes, properties and their instances. Given such an ontology, the OWL formal semantics specifies how to derive its logical consequences, i.e. facts not literally present in the ontology, but entailed by the semantics. These entailments may be based on a single document or multiple distributed documents that have been combined using defined OWL mechanisms".

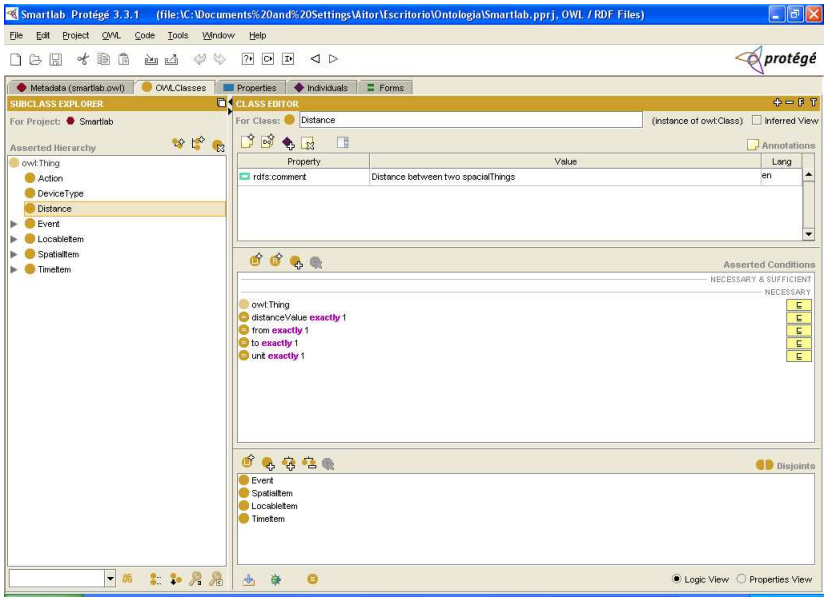

Fig. 2 Diseñando una ontología con Protégé

SPARQL

SPARQL es un lenguaje de consultas para RDF. Mediante SPARQL es posible recuperar la información de un grafo RDF de una manera sencilla expresando las consultas como triples. Al realizar una consulta mediante SPARQL se intentará buscar una coincidencia entre el patrón expresado mediante los triples y el grafo RDF, devolviendo los valores que se especifiquen con la sentencia SELECT. Un ejemplo de una consulta sencilla sería recuperar los nombres de todos los usuarios que estén dentro de una habitación en concreto:

```
SELECT ?name
WHERE
{
      ?p rdfs:type sl:person
      ?p sl:placedIn ?room
      ?room sl:name 'Smartlab'
      ?p sl:name ?name
}
```

Esta consulta devolvería los nombres de todos los elementos de tipo smartlab:person que se hayan en la habitación "Smartlab" existentes en el grado RDF:

```
_:a  smartlab:name "Iker" .
_:b  smartlab:name "Unai" .
_:c  smartlab:name "Aitor" .
_:d  smartlab:name "Xabi" .
_:e  smartlab:name "Pablo" .
```

PELLET

Pellet es un razonador OWL Open Source y gratuito desarrollado el Mindswap Lab de la Universidad de Maryland. En el desarrollo del sistema se ha utilizado para comprobar la consistencia de la ontología a la hora de diseñarla. Pellet proporciona las siguientes funcionalidades:

- Chequeo de consistencia de la ontología.
- Es posible comprobar si una clase puede ser satisfecha, es decir, puede tener instancias.
- Inferencia de nuevas clases.
- Clasificación automática de las instancias.
- Comprobación de la sintaxis en grafos RDF.

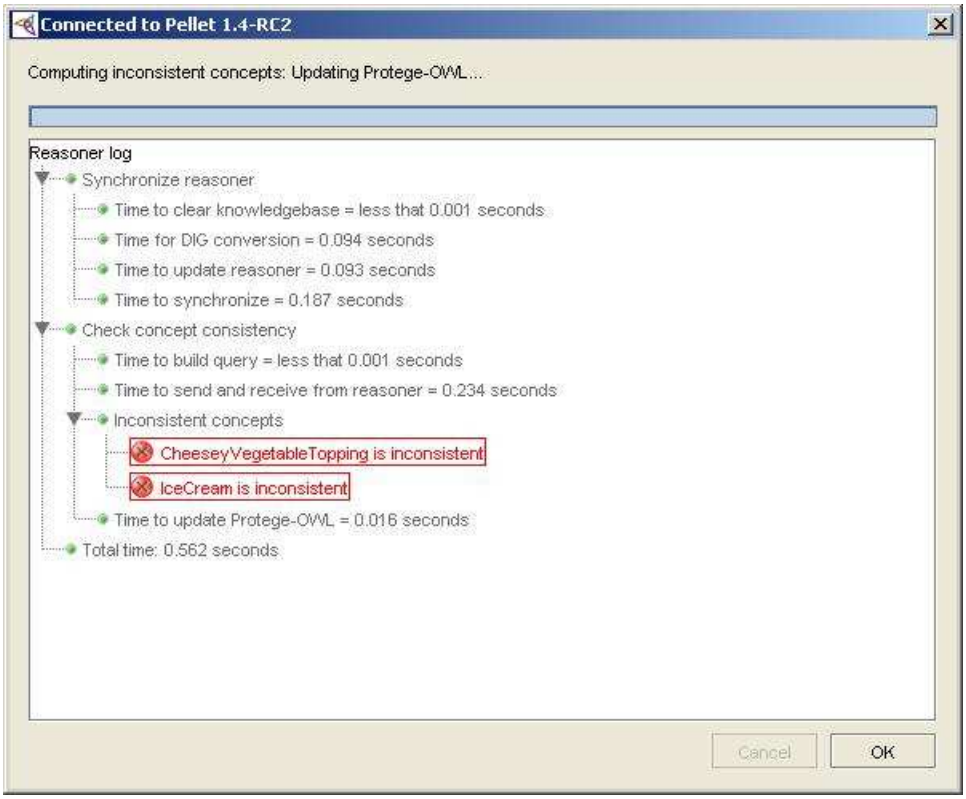

Fig. 3 Validando la ontología diseñada mediante Pellet

1. ARQUITECTURA

La arquitectura del sistema desarrollado se divide en tres bloques principales: el mecanismo de descubrimiento, los bundles de dispositivos y la capa de de inferencia semántica. Además el elemento central que mantiene la cohesión es la base de conocimiento (la ontología), donde se almacena toda la información.

Cuando un nuevo bundle de dispositivo es descubierto se le solicita su información de dominio y se añade esta información a la base de conocimiento y a las reglas de dominio. Puede suceder que las reglas de un dispositivo dependan de otro dispositivo diferente. Aunque en un principio un dispositivo no tendría por qué conocer al resto en este caso si se supone cierto conocimiento. Esto provoca cierto acoplamiento entre los diferentes bundles de dispositivo, aunque éste en mínimo.

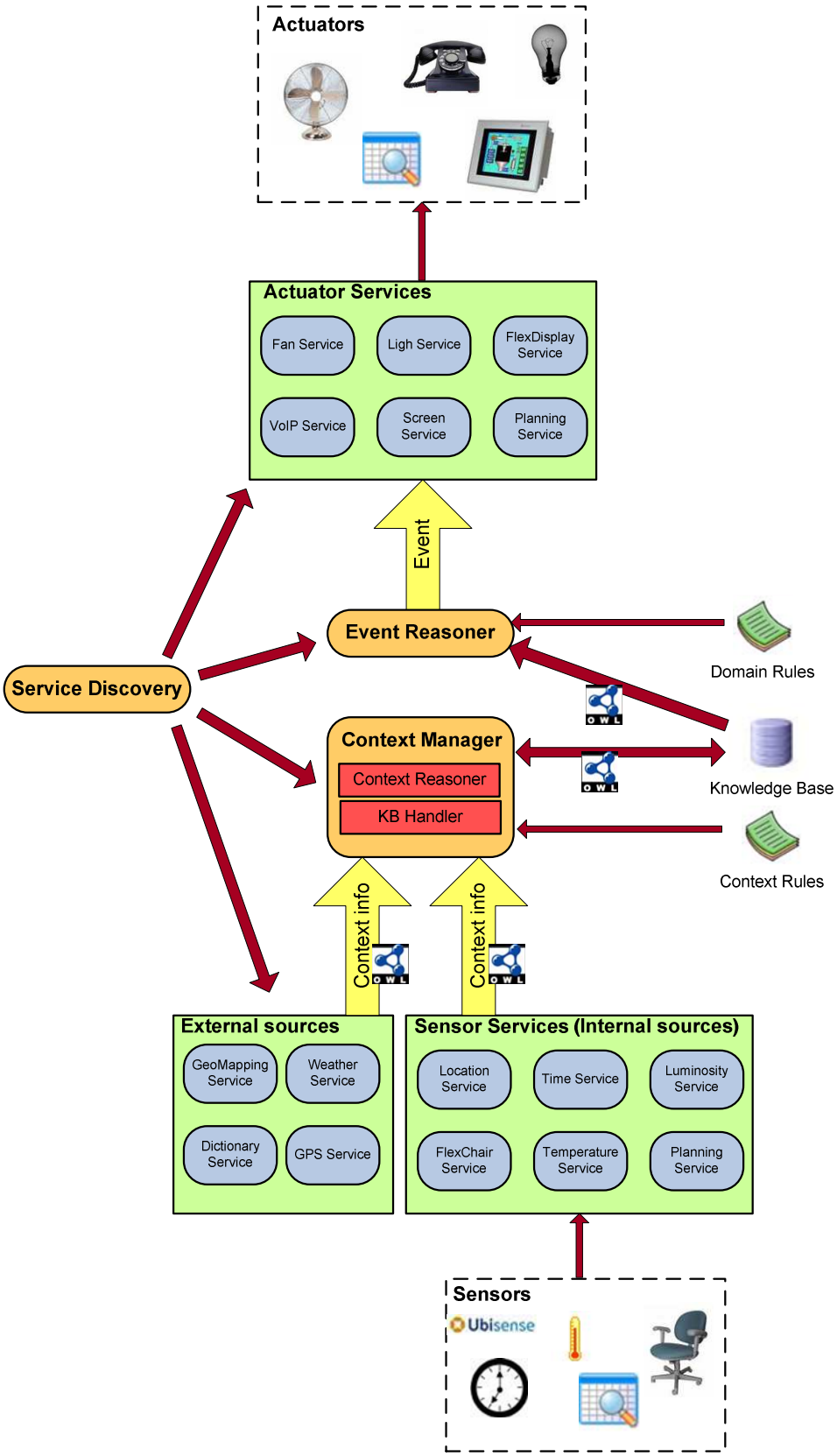

Fig. 4 Arquitectura del sistema

Cada uno de los elementos de la capa de inferencia tendrá un papel fundamental dentro del sistema, siendo sus tareas las siguientes:

- *Context Manager*: Los datos recogidos por los diferentes dispositivos no tienen en un principio ningún tipo de significado semántico, sólo son una serie de valores. La capa de semantización será la encargada de darle significado a esos valores y después los almacenará en la base de conocimiento. Un ejemplo de esto son los datos recogidos del sistema de localización. En su estado inicial los datos sólo son dos números reales pero el sistema de semantización los interpreta y les añade información semántica, indicando que son los valores X e Y, que se refieren a la posición de un objeto en concreto y que esos valores se miden desde un punto de referencia dado. Puesto que puede haber varios sistemas de localización con diferentes puntos de referencia, si no se le añadiera este tipo de información a los datos no sería posible procesarlos correctamente. Un ejemplo de información semantizada sería el siguiente:

```
<BinnaryLight rdf:ID="LightArea2">
<placedIn rdf:resource="#LockerRoom"/>
<name rdf:datatype="XMLSchema#string">
LightArea2
</name>
<x>23.3</x>
<y>45.5</y>
</BinnaryLight>
```

  En el ejemplo se puede ver que los valores (23.3,45.5) pertenecen al dispositivo *LighArea2* y toman como punto de referencia *LockerRoom*.

- *Context Reasoner y Event Reasoner*: El motor de inferencia se encarga de ampliar la información del contexto explicitando la información que se encuentra implícita. Para hacer esto hará uso tanto de las relaciones semánticas de la ontología como de reglas diseñadas especialmente para el dominio. Una vez haya inferido nueva información razonará cuales son los eventos del sistema: hay una reunión, hay una inundación, alguien ha entrado a una habitación etc…

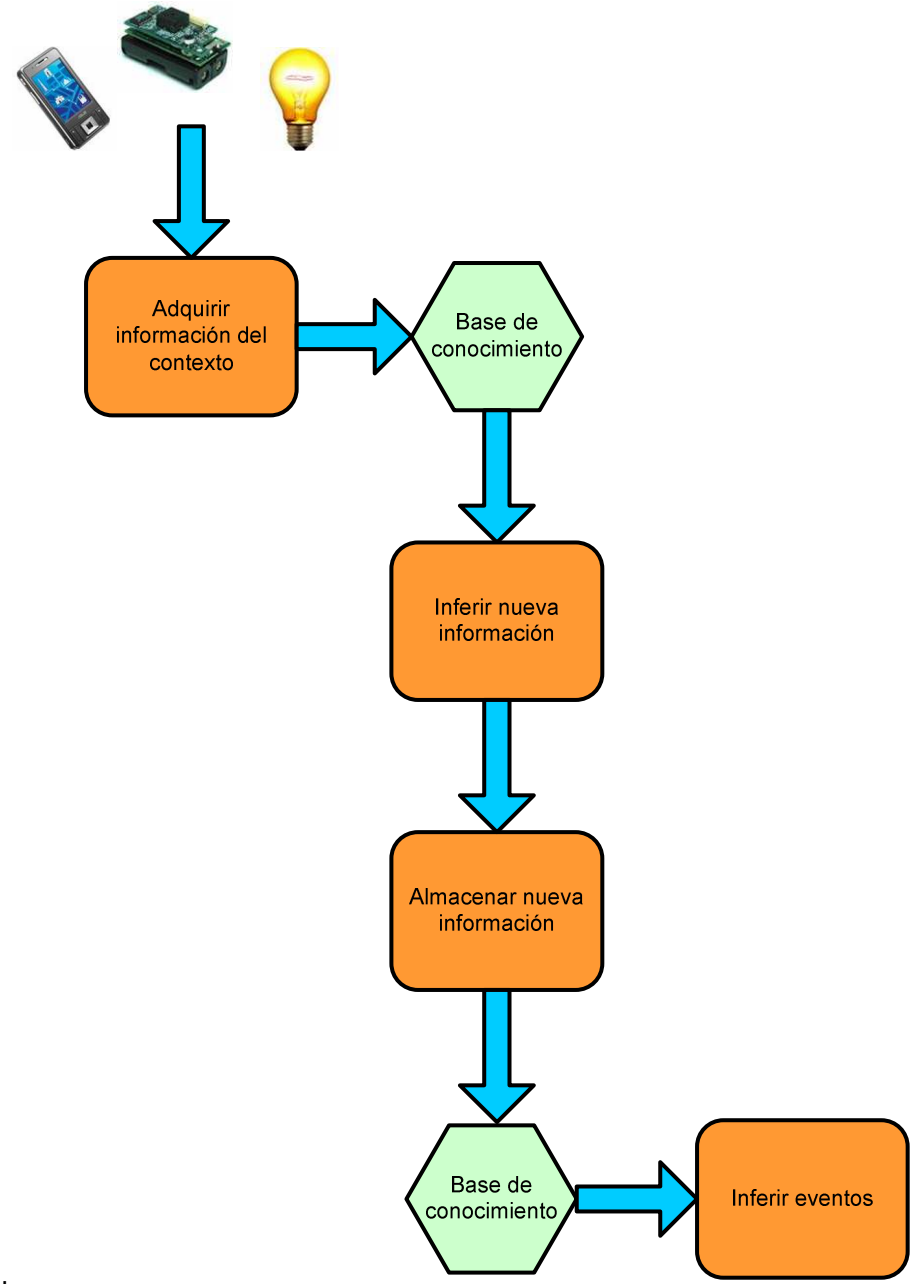

Fig. 5 Funcionamiento del motor de inferencia

3. DISEÑO DE LA ONTOLOGÍA

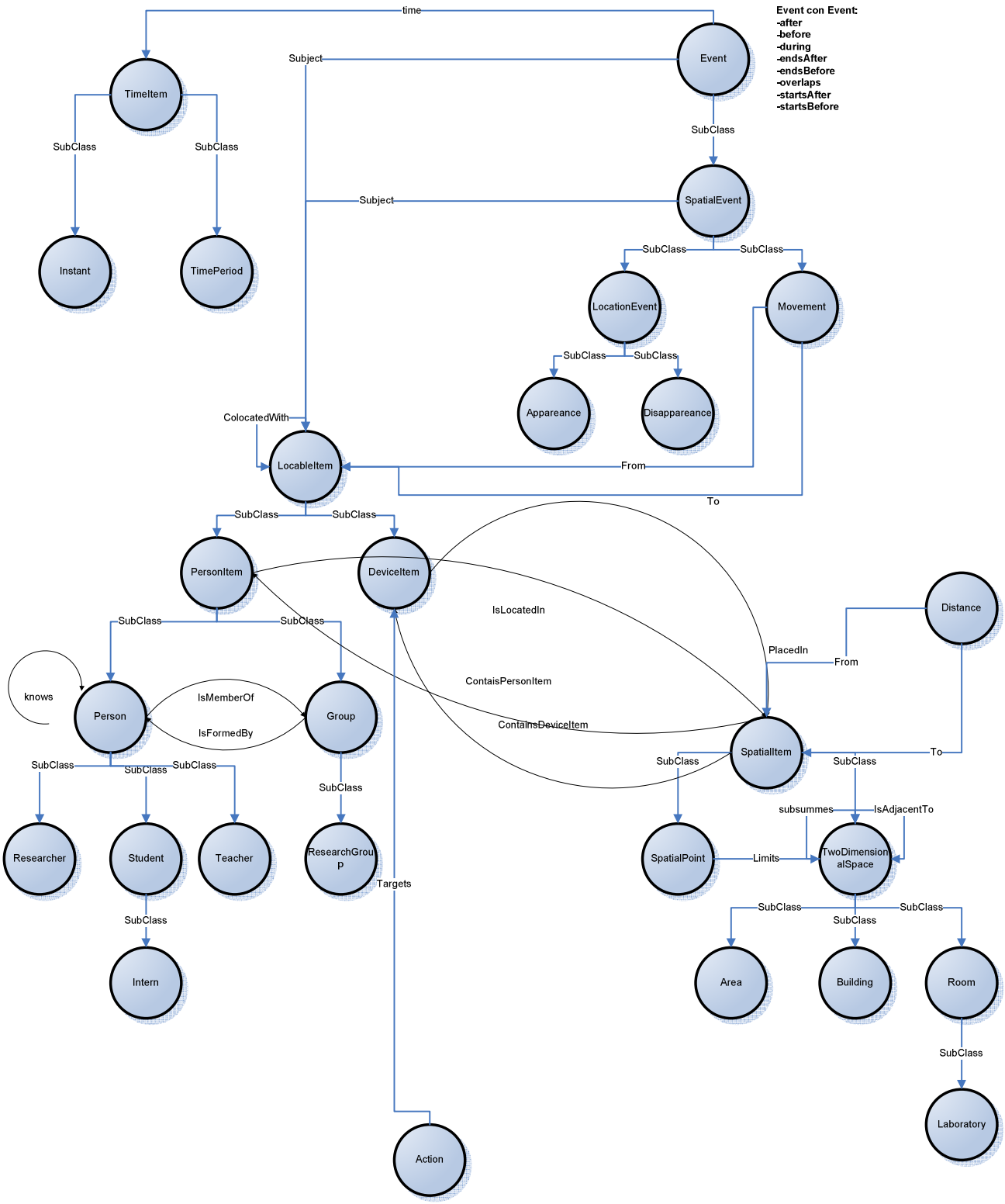

Fig. 6 Esquema de la ontología

3.1 TIMEITEM

Los elementos hijo de *TimeItem* permiten que la ontología modele el tiempo. Esto permitirá identificar cuáles son las relaciones temporales entre distintos eventos, pudiendo ordenarlos cronológicamente y facilitando la planificación de los mismos. El tiempo puede definirse de dos maneras. Como un instante mediante la clase *Instant*:

```
<owl:Class rdf:about="#Instant">
    <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
    >A time instant</rdfs:comment>
    <owl:disjointWith rdf:resource="#TimePeriod"/>
    <rdfs:subClassOf rdf:resource="#TimeItem"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:DatatypeProperty rdf:ID="value"/>
        </owl:onProperty>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

O como un periodo delimitado por dos instantes (*end* y *start*) mediante la clase *TimePeriod*:

```
<owl:Class rdf:ID="TimePeriod">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="TimeItem"/>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="end"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
<owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="start"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

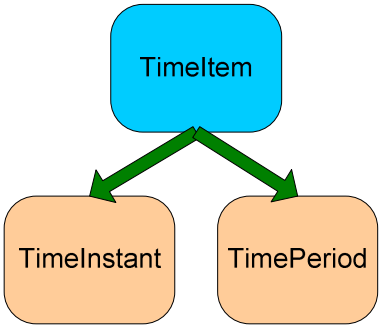

Fig. 6 Jerarquía de TimeItem

3.2 SPATIALITEM

Los elementos hijo de *SpatialItem* permiten que la ontología modele el espacio. Gracias a las relaciones espaciales el sistema podrá identificar que entidades se encuentran en la misma localización, calcular distancias y planificar rutas de movimiento. La jerarquía de *SpatialItem* es más compleja que la de *TimeItem*, teniendo dos elementos principales: los puntos y los espacios bidimensionales.

Los puntos representados por la clase *SpatialPoint* tienen propiedades para indicar su posición X e Y. Además tienen las siguientes propiedades que permiten modelar relaciones espaciales:

- *Limits*: Indica si forman parte de los limites de algún espacio bidimensional.
- *Contains*: Indica si alguna entidad está situada en ese punto.
- *subsummedBy*: Indica si el punto esta situado dentro de algún espacio bidimensional.

Los espacios bidimensionales se dividen en áreas, edificios y habitaciones. A su vez existe un tipo especializado de habitación, los laboratorios. Los espacios bidimensionales tienen las siguientes propiedades:

- *isAdjacent*: indica junto a que otros espacios se encuentra.
- *isLimitedBy*: indica los puntos que limitan el espacio.
- *contains*: indica que entidades contiene.
- *subsumes*: indica que otros espacios se encuentran dentro de él.
- *subsummedBy*: indica si se encuentra dentro de otro espacio.

Además de estas propiedades comunes los laboratorios tienen la propiedad *hasResearchers* que permite indicar que investigadores están asignados al mismo.

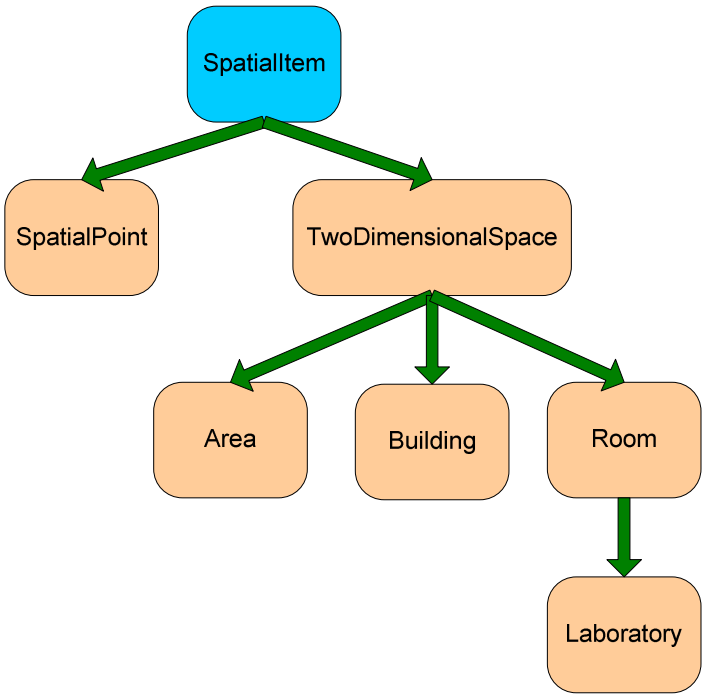

Fig. 7Jerarquía de SpatialItem

3.3 LocableItem

Los elementos hijo de *LocableItem* son aquellas entidades que tienen una localización espacial. Se dividen en dos categorías *PersonItem* (entidades relacionadas con las personas) y *DeviceItem* (dispositivos).

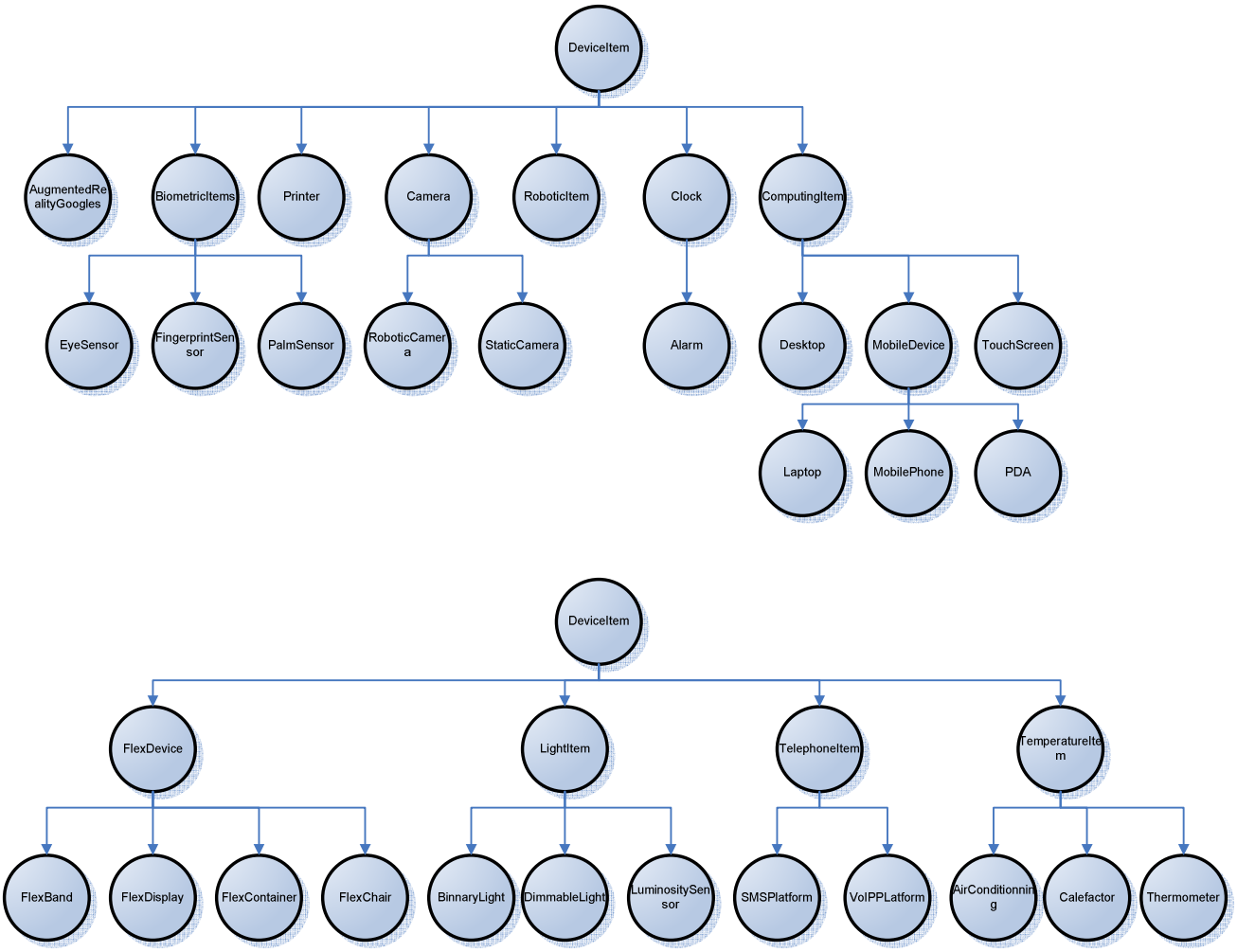

Fig. 8 Taxonomía de dispositivos

DeviceItem contiene todos los tipos de dispositivos que pueden existir en el sistema: dispositivos domóticos, cámaras, sensores, actuadores, etc… Los dispositivos tienen las siguientes propiedades:

- *deviceType*: Tipo del dispositivo.
- *Name*: nombre del dispositivo.
- *ID*: identificador del dispositivo.
- *PlacedIn*: indica el lugar donde se encuentra.
- *ColocatedWith*: indica que otras entidades se encuentran en el mismo lugar.

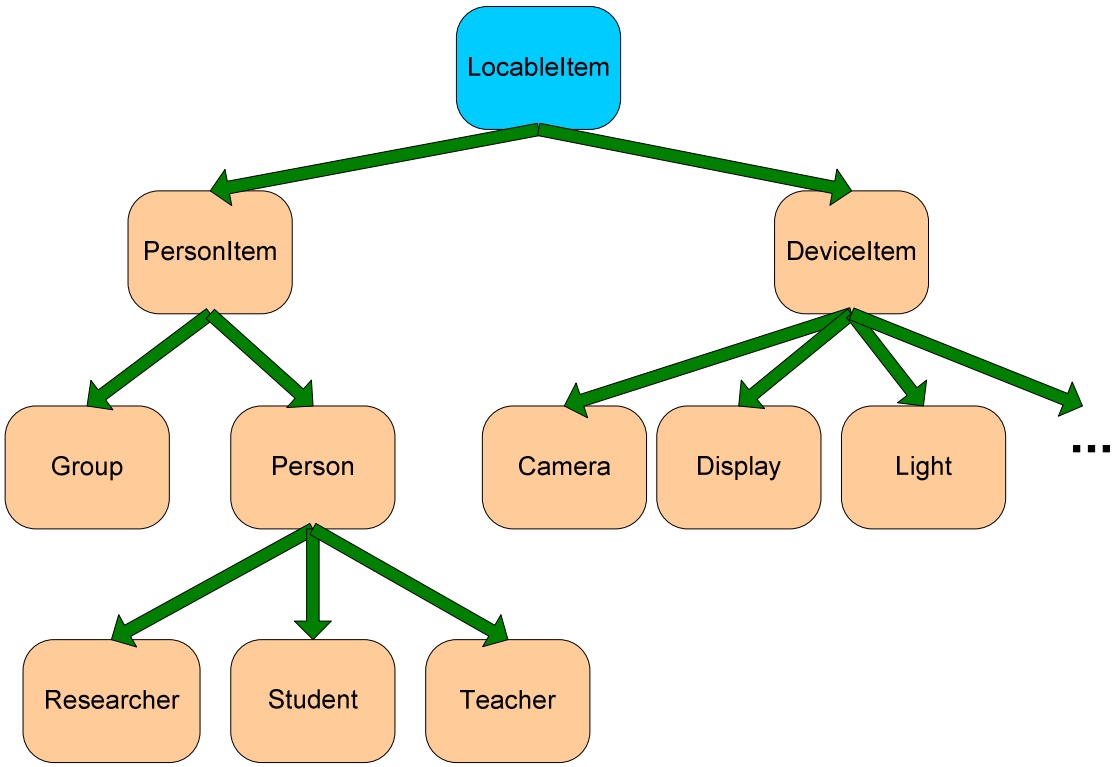

Fig. 8 Jerarquia de LocableItem

PersonItem se divide a su vez en dos categorías: grupos y personas. Un grupo esta formado por una serie de personas y tiene las siguientes categorías:

- *Name*: nombre del grupo.
- *IsFormedBy*: Personas que forman el grupo.
- *PlacedIn*: indica el lugar donde se encuentra. Todas las personas del grupo deben estar en el mismo lugar para que esta propiedad tenga algún valor asignado.
- *ColocatedWith*: indica que otras entidades se encuentran en el mismo lugar.

Las personas tienen una serie de propiedades que sirven para especificar sus datos (nombre, apellidos, dirección, teléfono, etc…). Además de las propiedades con los datos personales tienen también las siguientes:

- *IsMemberOf*: indica si es miembro de algún grupo.
- *Kwons*: indica a que otras personas conoce
- *PlacedIn*: indica el lugar donde se encuentra
- *ColocatedWith*: indica que otras entidades se encuentran en el mismo lugar.

Existen a su vez tres tipos especializados de personas: Profesores, estudiantes e investigadores.

3.4 EVENT

Las clases hijo de *Event* permiten modelas los sucesos del sistema. Un evento siempre esta ligado a un *TimeItem* (ya sea un instante o un periodo de tiempo) y puede tener asociado un *SpatialItem*. La clase *Event* tiene las siguientes propiedades:

- *After*: indica que sucede después de otro evento.
- *Before*: indica que eventos suceden antes.
- *During*: indica que eventos suceden dentro del periodo de tiempo del evento.
- *EndsAfter*: Indica que otro evento delimitado por un periodo acaba después.
- *EndsBefore*: Indica que otro evento delimitado por un periodo acaba antes.
- *Overlaps*: indica que el periodo de tiempo del evento se solapa con otros eventos.
- *StartsAfter*: Indica que otro evento delimitado por un periodo empieza después.
- *StartsBefore*: Indica que otro evento delimitado por un periodo empieza antes.
- *Time*: el *TimeItem* del evento.

Como se puede ver cuando varios eventos suceden en periodos de tiempo en vez de en instantes las relaciones temporales entre ellos pueden ser bastante complejas.

Además existe un tipo especializado de evento, el SpatialEvent. Este tipo de evento tiene una propiedad *location* que permite indicar donde ha sucedido. A su vez hay dos tipos especializados de *SpatialEvent* que tienen que ver con la localización de las diferentes entidades del sistema. No es realista esperar que todas las entidades tengan una posición estática que no cambie a lo largo del tiempo. Esto puede ser así con luces o cámaras de vigilancia pero es de esperar que las personas, dispositivos móviles, microbots, etc… se muevan por el edificio. Los eventos *Appareance*, *Disappareance* y *Movement* permiten dejar constancia de estos movimientos.

El evento *Movement* indica cuando un *LocableItem* se ha movido de una posición a otra. El evento *Appareance* indica que un *LocableItem* a aparecido en el sistema y que anteriormente no se conocía su localización (si se conociera sería un movimiento). El evento *Disappareance* indica que un *LocableItem* con localización conocida ya no puede ser encontrado.

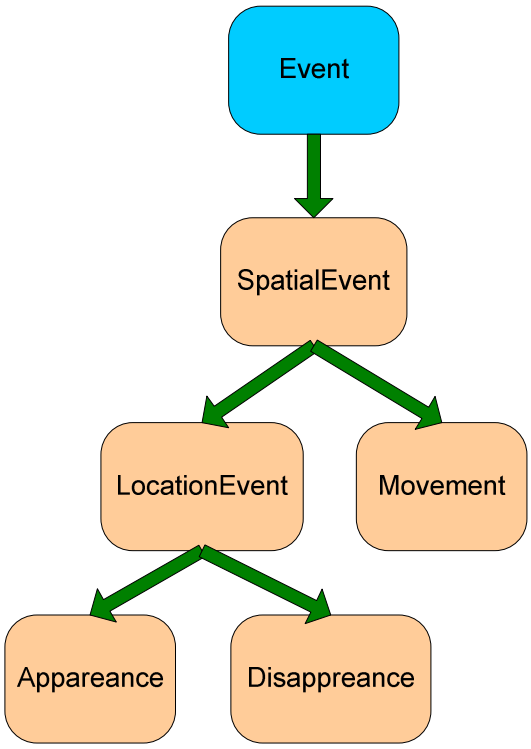

Fig. 9 Jerarquía de Event

4. MOTOR DE INFERENCIA

A la hora de diseñar el motor de inferencia se presentan tres opciones:

1. Tener un motor semántico y un motor de reglas separados.

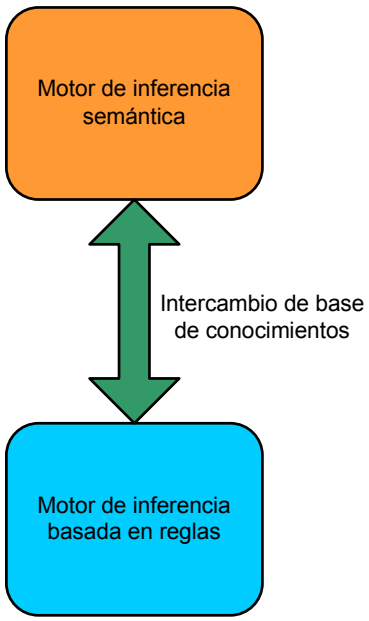

Fig. 10 Motor de reglas y semántico separados

2. Embeber el motor semántico dentro del motor de reglas.

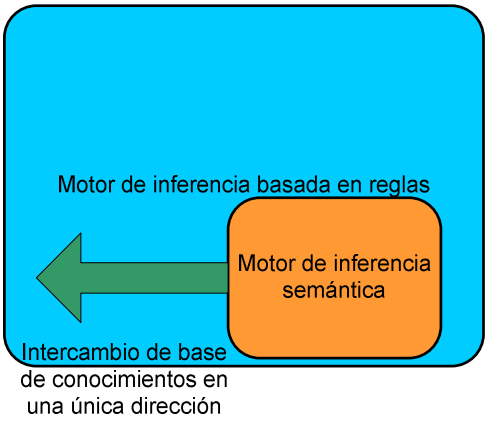

Fig. 11 Motor semántico embebido

3. Expresar la semántica de RDF y OWL como reglas y sólo utilizar el motor de reglas.

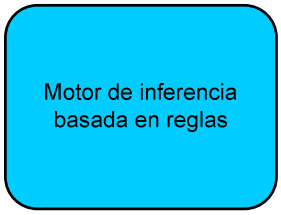

Fig. 12 Un único motor de reglas

Para decidir entre una de las tres opciones se han analizado las ventajas e inconvenientes de cada una de ellas, que se representan en la siguiente tabla:

Tabla 1 Comparación entre diseños de motores de inferencia

MOTOR	VENTAJAS	INCONVENIENTES
Motores separados	El motor semántico puede ser un motor externo como Pellet[PELL] o Racer[RACE], por lo que la inferencia semántica sería rápida y eficiente.	Habría que implementar manualmente el intercambio de la base de conocimiento entre ambos motores. Este intercambio sería necesario cada vez que la base de conocimiento sea modificada de alguna manera, lo que disminuiría el rendimiento del sistema.
Motor embebido	El intercambio de la base de conocimiento se haría de manera automática.	El intercambio de conocimiento sólo podría hacerse en una dirección, por lo que el motor semántico no podría observar los cambios hechos en la base de conocimiento por el motor de reglas. La eficiencia baja al estar un motor embebido en otro.
Sólo motor de reglas	No es necesario intercambio de la base de conocimiento. Se puede especificar que reglas semánticas que se deben tener en cuenta, pudiendo eliminar las que no sean necesarias en el sistema y aumentando el rendimiento.	Deben de especificarse las reglas semánticas manualmente.

Analizando la tabla se ha llegado a la conclusión que el diseño más adecuado para el sistema es el tercero, ya que habrá cambios frecuentes en la base de conocimiento que serán provocados tanto por la inferencia de las reglas como por la inferencia semántica. Además el posible inconveniente de la tercera opción de diseño se convierte en este caso en una ventaja ya que permite seleccionar que reglas semánticas va a implementar el sistema. De esta manera no se implementaran aquellas reglas que la ontología diseñada no vaya a utilizar por lo que el número de comprobaciones que realizará el motor será menor.

El motor de reglas utilizado será el propio motor de reglas incorporado en JENA, ya que tiene una serie de características que resultan necesarias para el sistema:

- Puede trabajar con una base de conocimientos expresada como una ontología.
- Permite expresar las reglas en formato de triples: (?a esUn Mamífero) -> (?a esUn Animal).

- Permite encaminamiento hacia delante, hacia atrás e híbrido.
- Permite introducir nuevos predicados en la ontología (la base de conocimiento) de manera sencilla.

El motor realizará tres labores de inferencia que se dividirán en dos fases. En una primera fase el motor extraerá el conocimiento implícito en la ontología y procesará sus relaciones semánticas para expandir la base de conocimientos. Después, haciendo uso de esta base de conocimientos expandida inferirá las acciones a llevar a cabo por el sistema.

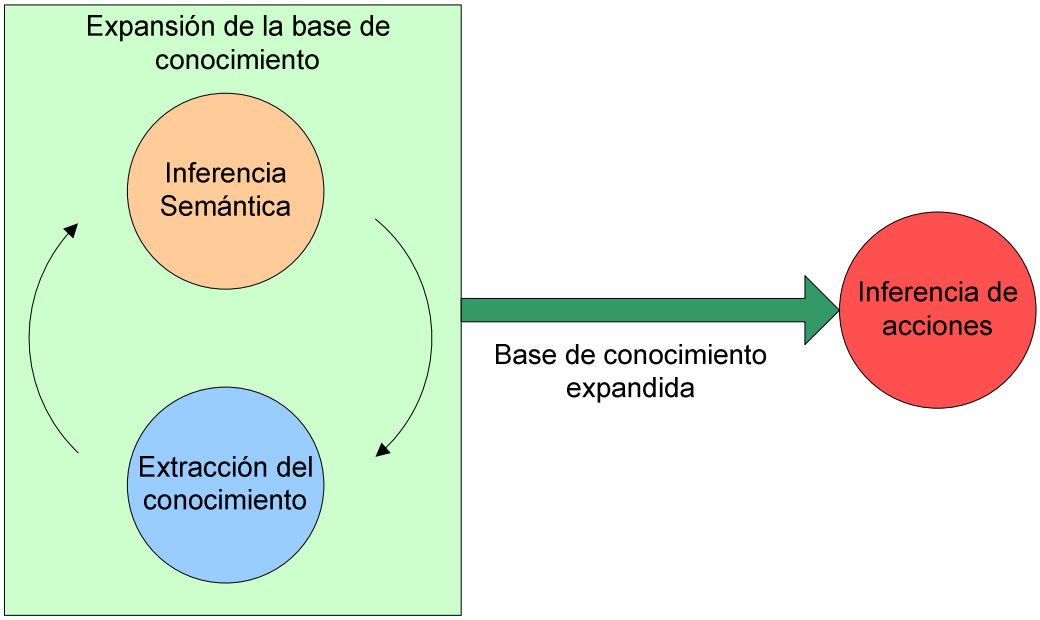

Fig. 13 Proceso de inferencia

6. REGLAS DEL SISTEMA

A la hora de diseñar las reglas del sistema se han especificado tres posibles categorías:

1. Reglas semánticas.
2. Reglas de extracción de conocimiento.
3. Reglas para la inferencia de eventos.

En la primera categoría se incluyen aquellas reglas que permiten expandir la base de conocimiento haciendo uso de las características de las relaciones semánticas. En la segunda categoría se encuentran aquellas reglas que son capaces de elicitar conocimiento de alto nivel a partir del conocimiento implícito y expandir la base de conocimiento. La última categoría contiene las reglas que infieren eventos a partir de la base de conocimiento.

6.1 REGLAS SEMÁNTICAS

Las reglas semánticas se dividen en dos subcategorías: RDF y OWL. Para poder implementar estas reglas se han analizado los modelos teóricos de ambas especificaciones y se han seleccionado las reglas necesarias para el sistema.

El modelo teórico de RDF [RDFTHEO] define doce reglas que se aplican de manera recursiva a un grafo RDF. Las reglas definidas son las siguientes (en formato de triples):

Tabla 2 Reglas semánticas para RDF

	IF	THEN
1a	xxx aaa yyy	xxx rdf:type rdfs:Resource
1b	xxx aaa yyy	aaa rdf:type rdf:Property
1c	xxx aaa uuu	uuu rdf:type rdfs:Resource
2	aaa rdfs:subPropertyOf bbb bbb rdfs:subPropertyOf ccc	aaa rdfs:subPropertyOf ccc
3a	aaa rdfs:subPropertyOf bbb	aaa rdf:type rdf:Property
3b	aaa rdfs:subPropertyOf bbb	bbb rdf:type rdf:Property
4	xxx aaa yyy aaa rdfs:subPropertyOf bbb	xxx bbb yyy
5	xxx aaa yyy	uuu rdf:type zzz

	aaa rdf:domain zzz	
6	xxx aaa uuu aaa rdf:range zzz	uuu rdf:type zzz
7	xxx rdf:type uuu	uuu rdf:type rdfs:Class
8	xxx rdf:type rdfs:Class	xxx rdfs:subClassOf rdfs:Resource
9a	xxx rdfs:subClassOf uuu	uuu rdf:type rdfs:Class
9b	xxx rdfs:subClassOf yyy.	xxx rdf:type rdfs:Class
10	xxx rdfs:subClassOf yyy yyy rdfs:subClassOf zzz	xxx rdfs:subClassOf zzz
11	xxx rdfs:subClassOf yyy aaa rdf:type xxx	aaa rdf:type yyy .
12	aaa rdf:type rdfs:ConstraintResource aaa rdf:type rdf:Property	aaa rdf:type rdfs:ConstraintProperty

La aplicación de estas reglas sobre un grafo RDF es un proceso finito que dará como resultado un grafo expandido. Por ejemplo si se le aplicaran esas reglas a un grafo con el triple "`Aitor esUna Persona`" el resultado sería:

```
Aitor esUna Persona
Aitor rdf:type rdfs:Resource
Persona rdf:type rdfs:Resource
esUna rdf:type rdf:Property
rdf:type rdf:type rdf:Property
rdf:Property rdf:type rdfs:Class
rdf:Property rdfs:subClassOf rdfs:Resource
```

De estas reglas no serán necesarias todas para el razonamiento semántico del sistema por lo tanto sólo se implementarán las reglas 2, 4, 5, 6, 10 y 11. Estas reglas serán traducidas al formato de reglas de JENA, por lo que quedarán expresadas de la siguiente manera:

```
[rdfs2:
      (?u rdfs:subPropertyOf ?v),
      (?v rdfs:subPropertyOf ?x)
      ->
      (?u rdfs:subPropertyOf ?x)
]
```

```
[rdfs4:
      (?a ?p ?b),
      (?p rdfs:subPropertyOf ?q)
      ->
      (?a ?q ?b)
]

[rdfs5:
      (?x ?p ?y),
      (?p rdfs:domain ?c)
      ->
      (?x rdf:type ?c)
]

[rdfs6:
      (?x ?p ?y),
      (?p rdfs:range ?c)
      ->
      (?y rdf:type ?c)
]

[rdfs10:
      (?u rdfs:subClassOf ?v),
      (?v rdfs:subClassOf ?x)
      ->
      (?u rdfs:subClassOf ?x)
]

[rdfs11:
      (?x rdfs:subClassOf ?y),
      (?a rdf:type ?x)
      ->
      (?a rdf:type ?y)
]
```

En cuanto a las reglas semánticas de OWL[OWLTHEO] sólo son utilizadas tres en la ontología para definir las características de algunas de las propiedades. Las reglas utilizadas son: la regla para definir propiedades transitivas, la regla para definir propiedades simétricas y la regla para definir propiedades inversas. Estas reglas son expresadas de la siguiente manera:

```
OWL-InverseOf:
      (?x ?prop1 ?y),
      (?prop1 owl:inverseOf ?prop2)
      ->
      (?y ?prop2 ?x)
]

 [OWL-Symmetric:
      (?x ?p ?y),
      (?p owl:inverseOf ?p)
      ->
      (?y ?p ?x)
```

```
]

[OWL-TransitiveProperty:
      (?x ?p ?y),
      (?y ?p z),
      (?p rdf:type owl:TransitiveProperty)
      ->
      (?x ?p ?z)
]
```

Estas reglas semánticas pueden ser utilizadas con cualquier ontología, ya que no dependen del sistema que se esté desarrollando.

6.2 REGLAS DE EXTRACCIÓN DEL CONTEXTO

Estas reglas permiten elicitar conocimiento de alto nivel a partir del conocimiento implícito en la ontología. Los tres aspectos más importantes del sistema desarrollado son el qué, el cuándo y el dónde. Por lo tanto el conocimiento de alto nivel inferido se basa en el tiempo y en el lugar. A partir de las reglas diseñadas es posible ordenar eventos temporalmente y conocer las relaciones espaciales entre diferentes entidades, por lo tanto estas reglas se dividen en dos categorías: inferencia temporal e inferencia espacial.

La inferencia temporal se encarga de hacer explícitas las relaciones temporales entre diferentes sucesos. Estas relaciones se encuentran implícitas en su timestamp, pero infiriéndolas se consigue una mayor facilidad al tratarlas. Un evento puede tener dos tipos de tiempos, un instante o un periodo (delimitado por dos instantes), que dependen de su duración. La primera regla temporal comprueba entre si eventos que han sucedió en un instante infiriendo cual ha sucedido antes:

```
[TEMPORAL1-eventInstantBefore:
      (?ev1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Event>),
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#time> ?ins1),
      (?ins1 <http://www.morelab.deusto.es/smartlab.owl#value> ?v1),
      (?ev2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Event>),
      (?ev2 <http://www.morelab.deusto.es/smartlab.owl#time> ?ins2),
      (?ins2 <http://www.morelab.deusto.es/smartlab.owl#value> ?v2),
      lessThan(?v1,?v2)
      ->
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#before> ?ev2)
]
```

La segunda regla comprueba si dos eventos ocurren en el mismo instante:

```
[TEMPORAL2-eventInstantDuring:
```

```
      (?ev1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Event>),
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#time> ?ins1),
      (?ins1 <http://www.morelab.deusto.es/smartlab.owl#value> ?v1),
      (?ev2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Event>),
      (?ev2 <http://www.morelab.deusto.es/smartlab.owl#time> ?ins2),
      (?ins2 <http://www.morelab.deusto.es/smartlab.owl#value> ?v2),
      (?ev1 owl:distinctMembers ?ev2)
      equal(?v1,?v2)
      ->
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#during> ?ev2)
]
```

Con estas dos reglas se cubren las posibilidades de eventos que suceden en un instante. No es necesaria una regla para definir si un evento ha ocurrido después de otro ya que la propiedad `<http://www.morelab.deusto.es/smartlab.owl#before>` es inversa a la propiedad `<http://www.morelab.deusto.es/smartlab.owl#after>` y por lo tanto es cubierta por las reglas semánticas de OWL.

La tercera regla comprueba si un evento que sucede durante un periodo comienza antes que otro:

```
[TEMPORAL3-eventPeriodStartsBefore:
      (?ev1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Event>),
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#time> ?per1),
      (?per1 <http://www.morelab.deusto.es/smartlab.owl#start> ?ins1),
      (?ins1 <http://www.morelab.deusto.es/smartlab.owl#value> ?start1),
      (?ev2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Event>),
      (?ev2 <http://www.morelab.deusto.es/smartlab.owl#time> ?per2),
      (?per2 <http://www.morelab.deusto.es/smartlab.owl#start> ?ins2),
      (?ins2 <http://www.morelab.deusto.es/smartlab.owl#value> ?start2),
      lessThan(?start1,?start2)
      ->
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#startsBefore> ?ev2)
]
```

La cuarta regla comprueba si un evento acaba que sucede durante un periodo antes que otro:

```
[TEMPORAL4-eventPeriodEndssBefore:
      (?ev1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Event>),
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#time> ?per1),
      (?per1 <http://www.morelab.deusto.es/smartlab.owl#end> ?ins1),
      (?ins1 <http://www.morelab.deusto.es/smartlab.owl#value> ?end1),
      (?ev2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Event>),
      (?ev2 <http://www.morelab.deusto.es/smartlab.owl#time> ?per2),
      (?per2 <http://www.morelab.deusto.es/smartlab.owl#end> ?ins2),
      (?ins2 <http://www.morelab.deusto.es/smartlab.owl#value> ?end2),
      lessThan(?end1,?end2)
      ->
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#endsBefore> ?ev2)
```

```
]
```

Una vez se han ejecutado las reglas temporales 3 y 4 es posible saber si dos eventos se solapan o si un evento es contenido por otro mediante las reglas temporales 5 y 6:

```
[TEMPORAL5-eventPeriodDuring:
      (?ev1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Event>),
      (?ev2 <http://www.morelab.deusto.es/smartlab.owl#time> ?per2),
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#startsAfter> ?ev2),
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#endsBefore> ?ev2),
      ->
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#during> ?ev2)
]

[TEMPORAL6-eventPeriodOverlaps:
(     ?ev1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Event>),
      (?ev2 <http://www.morelab.deusto.es/smartlab.owl#time> ?per2),
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#startsAfter> ?ev2),
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#endsAfter> ?ev2),
      ->
      (?ev1 <http://www.morelab.deusto.es/smartlab.owl#overlaps> ?ev2)
]
```

La inferencia espacial se encarga de hacer explícitas las relaciones espaciales entre diferentes entidades. Aunque en un principio sólo se conocen las posiciones X e Y de cada entidad es posible inferir que entidades estas situadas juntas o que entidades contienen a otras y declararlo de una manera explícita mediante relaciones en la ontología. La primera regla espacial comprueba si dos personas se encuentran en la misma habitación:

```
[SPATIAL1-PersonColocatedWithPersonInRoom:
      (?thg1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#PersonItem>),
      (?thg1 <http://www.morelab.deusto.es/smartlab.owl#isLocatedIn> ?loc1),
      (?loc1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Room>),
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#name> ?name1),
      (?thg2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#PersonItem>),
      (?thg2 <http://www.morelab.deusto.es/smartlab.owl#isLocatedIn> ?loc2),
      (?loc2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Room>),
      (?loc2 <http://www.morelab.deusto.es/smartlab.owl#name> ?name2),
      equal(?name1,?name2)
      ->
(?thg1 <http://www.morelab.deusto.es/smartlab.owl#colocatedWith> ?thg2)
]
```

La segunda regla comprueba si dos personas se encuentran en la misma área:

```
[SPATIAL2-PersonColocatedWithPersonInArea:
```

```
      (?thg1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#PersonItem>),
      (?thg1 <http://www.morelab.deusto.es/smartlab.owl#isLocatedIn> ?loc1),
      (?loc1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Area>),
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#name> ?name1),
      (?thg2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#PersonItem>),
      (?thg2 <http://www.morelab.deusto.es/smartlab.owl#isLocatedIn> ?loc2),
      (?loc2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Area>),
      (?loc2 <http://www.morelab.deusto.es/smartlab.owl#name> ?name2),
      equal(?name1,?name2)
      ->
(?thg1 <http://www.morelab.deusto.es/smartlab.owl#colocatedWith> ?thg2)
]
```

Por último la tercera regla comprueba si dos personas se encuentran en el mismo punto:

```
[SPATIAL3-PersonColocatedWithPersonInPoint:
      (?thg1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#PersonItem>),
      (?thg1 <http://www.morelab.deusto.es/smartlab.owl#isLocatedIn> ?loc1),
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#x> ?x1),
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#y> ?y1),
      (?thg2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#PersonItem>),
      (?thg2 <http://www.morelab.deusto.es/smartlab.owl#isLocatedIn> ?loc2),
      (?loc2 <http://www.morelab.deusto.es/smartlab.owl#x> ?x2),
      (?loc2 <http://www.morelab.deusto.es/smartlab.owl#y> ?y2),
      equal(?x1,?x2)
      equal(?y1,?y2)
      ->
      (?thg1 <http://www.morelab.deusto.es/smartlab.owl#colocatedWith> ?thg2)
]
```

Las reglas 4,5 y 6 son similares pero para objetos inanimados:

```
[SPATIAL4-ItemColocatedWithItemInRoom:
      (?thg1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#DeviceItem>),
      (?thg1 <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?loc1),
      (?loc1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Room>),
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#name> ?name1),
      (?thg2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#DeviceItem>),
      (?thg2 <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?loc2),
      (?loc2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Room>),
      (?loc2 <http://www.morelab.deusto.es/smartlab.owl#name> ?name2),
      equal(?name1,?name2)
      ->
      (?thg1 <http://www.morelab.deusto.es/smartlab.owl#colocatedWith> ?thg2)
]

[SPATIAL5-ItemColocatedWithItemInArea:
      (?thg1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#DeviceItem>),
      (?thg1 <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?loc1),
      (?loc1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Area>),
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#name> ?name1),
```

```
      (?thg2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#DeviceItem>),
      (?thg2 <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?loc2),
      (?loc2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#Area>),
      (?loc2 <http://www.morelab.deusto.es/smartlab.owl#name> ?name2),
      equal(?name1,?name2)
      ->
      (?thg1 <http://www.morelab.deusto.es/smartlab.owl#colocatedWith> ?thg2)
]

[SPATIAL6-ItemColocatedWithItemInPoint:
      (?thg1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#DeviceItem>),
      (?thg1 <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?loc1),
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#x> ?x1),
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#y> ?y1),
      (?thg2 rdf:type <http://www.morelab.deusto.es/smartlab.owl#DeviceItem>),
      (?thg2 <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?loc2),
      (?loc2 <http://www.morelab.deusto.es/smartlab.owl#x> ?x2),
      (?loc2 <http://www.morelab.deusto.es/smartlab.owl#y> ?y2),
      equal(?x1,?x2)
      equal(?y1,?y2)
      ->
      (?thg1 <http://www.morelab.deusto.es/smartlab.owl#colocatedWith> ?thg2)
]
```

Las reglas 7 y 8 comprueban las relaciones entre objetos y personas y son similares a las anteriores reglas. Todas las anteriores reglas espaciales dan por hecho que se conoce la habitación o área en la que se encuentran las entidades. Para que esto sea posible es necesario comprobar si la posición X e Y de una entidad esta contenida por algún espacio bidimensional. Esto se logra con la regla 9:

```
[SPATIAL9-pointInTwoDimensionalSpace:
      (?loc1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#SpatialPoint>),
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#x> ?x),
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#y> ?y),
      (?area1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#TwoDimensionalSpace>),
      (?area1 <http://www.morelab.deusto.es/smartlab.owl#isLimitedBy> ?p1),
      (?area1 <http://www.morelab.deusto.es/smartlab.owl#isLimitedBy> ?p2),
      (?area1 <http://www.morelab.deusto.es/smartlab.owl#isLimitedBy> ?p3),
      (?area1 <http://www.morelab.deusto.es/smartlab.owl#isLimitedBy> ?p4),
      (?p1 <http://www.morelab.deusto.es/smartlab.owl#x> ?p1x),
      (?p1 <http://www.morelab.deusto.es/smartlab.owl#y> ?p1y),
      (?p2 <http://www.morelab.deusto.es/smartlab.owl#x> ?p2x),
      (?p2 <http://www.morelab.deusto.es/smartlab.owl#y> ?p2y),
      (?p3 <http://www.morelab.deusto.es/smartlab.owl#x> ?p3x),
      (?p3 <http://www.morelab.deusto.es/smartlab.owl#y> ?p3y),
      (?p4 <http://www.morelab.deusto.es/smartlab.owl#x> ?p4x),
      (?p4 <http://www.morelab.deusto.es/smartlab.owl#y> ?p4y),
      greaterThan(?x, ?p1x),
      greaterThan(?y, ?p1y),
      lessThan(?x,?p3x),
      lessThan(?y,?p3y),
      greaterThan(?x, ?p2x),
      lessThan(?y,?p2y),
```

```
      greaterThan(?y, ?p4y),
      lessThan(?x,?p4x),
      ->
      (?loc1 <http://www.morelab.deusto.es/smartlab.owl#isContainedBy> ?area1)
]
```

Por último las reglas 10 y 11 hacen uso de la información inferida por la regla 9 para comprobar donde se encuentran las entidades:

```
[SPATIAL10-PersonItemIsLocatedIn:
(?thg1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#PersonItem>),
(?thg1 <http://www.morelab.deusto.es/smartlab.owl#isLocatedIn> ?loc1),
(?loc1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#SpatialPoint>),
(?loc1 <http://www.morelab.deusto.es/smartlab.owl#isContainedBy> ?area1)
      ->
(?thg1 <http://www.morelab.deusto.es/smartlab.owl#isLocatedIn> ?area1)
]

[SPATIAL11-DeviceItemPlacedIn:
(?thg1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#DeviceItem>),
(?thg1 <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?loc1),
(?loc1 rdf:type <http://www.morelab.deusto.es/smartlab.owl#SpatialPoint>),
(?loc1 <http://www.morelab.deusto.es/smartlab.owl#isContainedBy> ?area1)
      ->
(?thg1 <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?area1)
]
```

Dado que los conceptos de tiempo y localización se pueden encontrar en la mayoría de aplicaciones de Inteligencia Ambiental estas reglas pueden adaptarse a otro sistema haciendo los cambios necesarios en caso de utilizar una ontología diferente.

6.3 REGLAS PARA LA INFERENCIA DE EVENTOS

El último tipo de reglas son aquellas utilizadas para inferir eventos a partir del contexto. Estas reglas se basan en el contexto extraído por las anteriores reglas para deducir que eventos deben lanzarse. Aunque las anteriores reglas son extrapolables a la mayoría de aplicaciones de inteligencia ambiental con algunos cambios las reglas de esta categoría dependen del sistema que se este desarrollando, ya que ellas serán las encargadas de definir su comportamiento. Para Smartlab se han definido 3 categorías: Reglas de reunión, reglas de inundación y reglas de seguridad.

Las reglas de reunión permiten identificar cuando acontece una reunión y preparar la habitación para la misma. Una reunión comienza cuando tres o mas personas se juntan en un área de reunión:

```
[EVENT-Meeting:
(?meetingArea rdf:type <http://www.morelab.deusto.es/smartlab.owl#MeetingArea>),
(?meetingArea <http://www.morelab.deusto.es/smartlab.owl#containsPersonItem> ?p1),
(?meetingArea <http://www.morelab.deusto.es/smartlab.owl#containsPersonItem> ?p2),
(?meetingArea <http://www.morelab.deusto.es/smartlab.owl#containsPersonItem> ?p3),
(?p1 <http://www.morelab.deusto.es/smartlab.owl#name> ?name1),
(?p2 <http://www.morelab.deusto.es/smartlab.owl#name> ?name2),
(?p3 <http://www.morelab.deusto.es/smartlab.owl#name> ?name3),
notEqual(?name1, ?name2),
notEqual(?name1, ?name2),
notEqual(?name1, ?name3),
notEqual(?name2, ?name3),
makeTemp(?meetingEvent)
->
(?meetingEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#LocableMeetingEvent>),
(?meetingEvent <http://www.morelab.deusto.es/smartlab.owl#place> ?meetingArea)
]
```

Si hay una reunión se encenderán las luces y las calefacciones de la habitación y se enviará un SMS al resto de personas avisándoles de la reunión:

```
[EVENT-SwitchOnLightsForMeeting:
(?meetingEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#LocableMeetingEvent>),
(?meetingEvent <http://www.morelab.deusto.es/smartlab.owl#place> ?nameArea)
(?light rdf:type <http://www.morelab.deusto.es/smartlab.owl#LightItem>),
(?light <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?nameArea),
makeTemp(?lightOnEvent)
->
(?lightOnEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#LightOnEvent>),
(?lightOnEvent <http://www.morelab.deusto.es/smartlab.owl#subject> ?light),
(?lightOnEvent <http://www.morelab.deusto.es/smartlab.owl#place> ?nameArea)
]

[EVENT-SwitchOnTempForMeeting:
(?meetingEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#LocableMeetingEvent>),
(?meetingEvent <http://www.morelab.deusto.es/smartlab.owl#place> ?nameArea)
(?calef rdf:type <http://www.morelab.deusto.es/smartlab.owl#Calefactor>),
(?calef <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?nameArea),
makeTemp(?tempEvent)
->
(?tempEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#CalefactorOnEvent>),
(?tempEvent <http://www.morelab.deusto.es/smartlab.owl#subject> ?calef),
(?tempEvent <http://www.morelab.deusto.es/smartlab.owl#place> ?nameArea)
]

[EVENT-SendMeetingSMS:
(?meetingEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#LocableMeetingEvent>),
(?meetingEvent <http://www.morelab.deusto.es/smartlab.owl#place> ?area1)
(?area1 <http://www.morelab.deusto.es/smartlab.owl#name> ?name1),
(?p1 <http://www.morelab.deusto.es/smartlab.owl#isLocatedIn> ?area2),
(?area2 <http://www.morelab.deusto.es/smartlab.owl#name> ?name2),
notEqual(?name1, ?name2),
(?p1 <http://www.morelab.deusto.es/smartlab.owl#mobile> ?mobile),
makeTemp(?smsEvent)
```

```
->
(?smsEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#SMSEvent>),
(?smsEvent <http://www.morelab.deusto.es/smartlab.owl#place> ?nameArea),
(?smsEvent <http://www.morelab.deusto.es/smartlab.owl#mobile> ?mobile)
]
```

Si se detecta una inundación se realizarán dos acciones, primero se apagarán todos los aparatos eléctricos de la habitación y se avisará de la misma mediante los FlexDisplay:

```
[EVENT-FloodSwitchOff:
(?sensor rdf:type <http://www.morelab.deusto.es/smartlab.owl#SensorValue>),
(?sensor <http://www.morelab.deusto.es/smartlab.owl#name> ?nameSensor),
equal(?nameSensor, 'flood'),
(?sensor <http://www.morelab.deusto.es/smartlab.owl#place> ?area),
(?device rdf:type <http://www.morelab.deusto.es/smartlab.owl#DeviceItem>),
(?device <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?area),
makeTemp(?floodEvent)
->
(?floodEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#SwitchOffEvent>),
(?floodEvent <http://www.morelab.deusto.es/smartlab.owl#place> ?nameArea),
(?floodEvent <http://www.morelab.deusto.es/smartlab.owl#subject> ?device)
]

[EVENT-FloodAlert:
(?sensor rdf:type <http://www.morelab.deusto.es/smartlab.owl#SensorValue>),
(?sensor <http://www.morelab.deusto.es/smartlab.owl#name> ?nameSensor),
equal(?nameSensor, 'flood'),
(?sensor <http://www.morelab.deusto.es/smartlab.owl#place> ?area),
makeTemp(?floodEvent)
->
(?floodEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#FloodAlertEvent>),
(?floodEvent <http://www.morelab.deusto.es/smartlab.owl#place> ?nameArea)
]
```

Por último cada vez que una ventana o una puerta sean abiertas las cámaras de la habitación capturarán una imagen de la persona que está entrando:

```
[EVENT-ScreenshotDoor:
(?sensor rdf:type <http://www.morelab.deusto.es/smartlab.owl#SensorValue>),
(?sensor <http://www.morelab.deusto.es/smartlab.owl#name> ?nameSensor),
equal(?nameSensor, 'door'),
(?sensor <http://www.morelab.deusto.es/smartlab.owl#place> ?area),
(?device rdf:type <http://www.morelab.deusto.es/smartlab.owl#Camera>),
(?device <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?area),
makeTemp(?camEvent)
->
(?camEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#SnapShootEvent>),
(?camEvent <http://www.morelab.deusto.es/smartlab.owl#subject> ?device)
```

```
]

[EVENT- ScreenshotWindow:
(?sensor rdf:type <http://www.morelab.deusto.es/smartlab.owl#SensorValue>),
(?sensor <http://www.morelab.deusto.es/smartlab.owl#name> ?nameSensor),
equal(?nameSensor, 'window'),
(?sensor <http://www.morelab.deusto.es/smartlab.owl#place> ?area),
(?device rdf:type <http://www.morelab.deusto.es/smartlab.owl#Camera>),
(?device <http://www.morelab.deusto.es/smartlab.owl#placedIn> ?area),
makeTemp(?camEvent)
->
(?camEvent rdf:type <http://www.morelab.deusto.es/smartlab.owl#SnapShootEvent>),
(?camEvent <http://www.morelab.deusto.es/smartlab.owl#subject> ?device)
]
```

REFERENCIAS

[CODA] The CoDAMoS Project: Context-Driven Adaptation of Mobile Services. Online, accedida el 17/may/2007. URL: http://www.cs.kuleuven.ac.be/distrinet/projects/CoDAMoS/ (2003).

[COBRA] H. Chen. An Intelligent Broker Architecture for Pervasive Context-Aware Systems. PhD thesis, University of Maryland, Baltimore County, 2004.

[CONON] Wang XH, Zhang DQ, Gu T, Pung HK. Ontology Based Context Modeling and Reasoning using OWL. Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops, 2004.

[SOCAM] Tao Gu, Hung Keng Pung, Da Qing Zhang. Toward an OSGi-Based Infrastructure for Context-Aware Applications. Pervasive Computing, 2004.

[PROT] The Protégé Ontology Editor and Knowledge Acquisition System. ONLINE. Accedida el 15/7/2007. URL: http://protege.stanford.edu/

[OKBC] Open Knowledge Base Connectivity. ONLINE. Accedida el 15/7/2007. URL: http://www.ai.sri.com/~okbc/

[JENA] Jena Semantic Web Framework. ONLINE. Accedida el 15/7/2007. URL: http://jena.sourceforge.net/

[SPAR] SPARQL query language. ONLINE. Accedida el 15/7/2007. URL: http://www.w3.org/TR/rdf-sparql-query/

[RDF] Resource Description Framework. ONLINE. Accedida el 15/7/2007. URL: http://www.w3.org/RDF/

[OWL] OWL Web Ontology Language. SPARQL query language. ONLINE. Accedida el 15/7/2007. URL: http://www.w3.org/TR/owl-features/

[OWLGUI] OWL Web Ontology Language Guide. SPARQL query language. ONLINE. Accedida el 15/7/2007. URL: http://www.w3.org/TR/owl-guide/

[PELL] Pellet: Motor de inferencia semántico. ONLINE. Accedida el 10/7/2007. URL: http://pellet.owldl.com/.

[RACE] Volker Haarslev, Ralf Möller. RACER System Description. Lecture Notes In Computer Science; Vol. 2083, pg 701-706. 2001.

[RDFTHEO] RDF Model Theory. ONLINE. Accedida el 10/7/2007. URL: http://www.w3.org/TR/2001/WD-rdf-mt-20010925/

[OWLTHEO] OWL Model Theory. ONLINE. Accedida el 10/7/2007. URL: http://www.w3.org/TR/2002/WD-owl-semantics-20021108/

[OWLS] OWL-S: Semantic Markup for Web Services ONLINE. Accedida el 10/7/2007. URL: http://www.w3.org/Submission/OWL-S/