Deliverable D5.4:

Integration of the open source catalogue with the semantic discovery engine – Version 2

Date: 28th December 2011
Author(s): Iker Larizgoitia (UIBK), Ioan Toma (UIBK)
Dissemination level: PU
WP: 5
Version: 1.0
Keywords: Catalogue, OGC, semantics, discovery
Description: Report presenting the integration strategies and interfaces of the open-source catalogue with the semantic discovery engine.

ICT for Environmental Services and Climate Change Adaption

Small or Medium-scale Focused Research Project
ENVISION (Environmental Services Infrastructure with Ontologies)
Project No: 249120
Project Runtime: 01/2010 – 12/2012
Copyright © ENVISION Consortium 2009-2012
Document metadata

Quality assurors and contributors

| Quality assuror(s) | Georgios Athanasopoulos (NKUA), Sébastien Mosser (SINTEF) |

Version history

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>09.12.2011</td>
<td>First draft for internal peer review.</td>
</tr>
<tr>
<td>0.7</td>
<td>19.12.2011</td>
<td>Comments from reviewers implemented.</td>
</tr>
<tr>
<td>0.8</td>
<td>23.12.2011</td>
<td>Technical review performed</td>
</tr>
<tr>
<td>1.0</td>
<td>28.12.2011</td>
<td>Final version.</td>
</tr>
</tbody>
</table>
Executive Summary

This document presents the final release of the integration of the semantic discovery following the OGC CSW standards. The resulting artefact, the Semantic Catalogue, provides discovery functionality taking advantage of semantic annotations and descriptions provided with the OGC services, leading to a more powerful and precise discovery service.

This document builds on the architecture and implementation presented in the first year release of the Semantic Catalogue. It provides an overview of the implemented features and describes the architecture, with emphasis on its components, their responsibilities and also detailed descriptions of the implementation.

The connection to other components of the ENVISION platform is also explained, mainly the WSML Goal editor and the Resource Module.

Finally, some conclusions are presented, highlighting the achievements and lessons learnt from the whole process of the development of the Semantic Catalogue.
Table of contents

EXECUTIVE SUMMARY .................................................................................................................. 3
TABLE OF CONTENTS ..................................................................................................................... 4
TABLE LIST ....................................................................................................................................... 5
FIGURE LIST ..................................................................................................................................... 5
1 INTRODUCTION ............................................................................................................................ 6
2 SEMANTIC CATALOGUE ................................................................................................................. 7
  2.1 SEMANTIC SERVICE DESCRIPTIONS ...................................................................................... 7
    2.1.1 WPS example ....................................................................................................................... 9
  2.2 SEMANTIC DISCOVERY AND PUBLICATION FUNCTIONALITY ............................................... 13
    2.2.1 Publication of semantically annotated resources .............................................................. 13
    2.2.2 Semantic discovery of services registered in the OGC catalogue ........................................ 15
3 SEMANTIC CATALOGUE ARCHITECTURE ................................................................................. 17
  3.1 DISCOVERY MODULE (USER INTERFACE) ........................................................................... 18
  3.2 SEMANTIC DISCOVERY SERVICE ......................................................................................... 18
  3.3 REASONER ............................................................................................................................. 19
    3.3.1 RDF Repository .................................................................................................................. 20
    3.3.2 Semantic Reasoner ............................................................................................................. 20
  3.4 OGC CATALOGUE ................................................................................................................... 20
    3.4.1 CSW access library ............................................................................................................. 20
  3.5 INTEGRATION WITH OTHER COMPONENTS ......................................................................... 21
    3.5.1 WSML Goal editor ............................................................................................................ 21
    3.5.2 Resource Module ............................................................................................................. 22
4 SEMANTIC CATALOGUE IMPLEMENTATION ........................................................................... 23
  4.1 DISCOVERY MODULE (USER INTERFACE) ........................................................................... 25
  4.2 SEMANTIC DISCOVERY SERVICE ......................................................................................... 27
  4.3 REASONER ............................................................................................................................. 29
    4.3.1 RDF Repository .................................................................................................................. 29
    4.3.2 Semantic Reasoner ............................................................................................................. 29
  4.4 OGC CATALOGUE ................................................................................................................... 30
    4.4.1 GeoNetwork ...................................................................................................................... 30
    4.4.2 CSW access library ............................................................................................................. 31
  4.5 INTEGRATION WITH OTHER COMPONENTS ......................................................................... 32
    4.5.1 WSML Goal Editor ............................................................................................................ 32
    4.5.2 Resource Module ............................................................................................................. 33
5 CONCLUSIONS .............................................................................................................................. 34

ANNEX ........................................................................................................................................... 35

WRAPPING A GWT INTERFACE IN A PORTLET .............................................................................. 35
CLIENT-SIDE INTER-PORTLET COMMUNICATION WITH SIMPLE JAVASCRIPT ............................... 36
REFERENCES ..................................................................................................................................... 38

ACRONYMS ....................................................................................................................................... 39
Table 1.- Discovery module visual components implementation.................. 18
Table 2.- HTTP bindings for the OGC Catalogue requests .......................... 19

Figure list

Figure 1.- Relations between the descriptions involved in the semantic discovery. ......................................................................................................................... 8
Figure 2 .- Publication diagram ................................................................................................................................. 15
Figure 3.- Semantic discovery diagram ................................................................. 16
Figure 4.- Semantic discovery architecture overview, second year release ...... 17
Figure 5.- WSML goal editor UI ......................................................................... 21
Figure 6.- Resource Module UI ........................................................................... 22
Figure 7.- Component diagram of the Semantic Catalogue ............................ 24
Figure 8.- Discovery Module UI ........................................................................ 27
Figure 9.- Semantic discovery service class diagram .................................... 28
Figure 10.- Reasoner class diagram .................................................................. 29
Figure 11.- CSW Access Library class diagram ............................................. 32
1 Introduction

The ENVIronmental Services Infrastructure with ONtologies project provides an environment for supporting non-ICT-skilled users in the process of semantic discovery, adaptive chaining and composition of environmental services. Semantic discovery is a key component of the infrastructure enabling users to find relevant OGC resources and services. It provides an intelligent and precise discovery mechanism in ENVISION as part of what we call the Semantic Catalogue. The Semantic Catalogue provides a semantic extension to standard OGC discovery of services, which uses semantic annotations and reasoning over service descriptions formalized by means of logics.

Work performed in WP5 was oriented towards the realization of our view of a Semantic Catalogue. In summary, in D5.1 [1] we analysed the state of the art in already existing OGC catalogue implementations. In D5.2 [2], we described the logical foundations for realizing semantic discovery using query containment algorithms and WSML descriptions of the services. In D5.3 [3] we presented the first release of our Semantic catalogue, which covered the standard OGC discovery inside the ENVISION platform, but the semantic mechanisms were still not fully integrated.

In this deliverable we present our second year release of the Semantic catalogue. We build our work here on top of all the previous deliverables from WP5 and describe the advances we made in the implementation of the Semantic catalogue. In D5.2 and D5.3, the foundations of the architecture were defined. This deliverable reports on the consolidation of the architecture (after applying the changes we considered necessary) and the implementation strategies and final functionality present in the Semantic Catalogue.

At this point of the development the integration with other partners and work package outcomes (especially WP4 and WP6) has also been crucial for the completion of the Semantic Catalogue functionality.

The following part of this deliverable is structured as follows: Section 2 presents a high-level overview of the Semantic catalogue, focusing on the semantic service descriptions, the discovery algorithm and the main operation of the Semantic Catalogue. Section 3 goes one step further and presents the architecture of the Semantic Catalogue and the responsibilities and connections between the main components that have been defined. Section 4 focuses on the implementation, giving additional low-level details on each component and finally Section 5 summarizes our work in the Semantic Catalogue and presents the final remarks.

During the development of the current release of the Semantic Catalogue we have faced different implementation and technical problems. Some of them we think are relevant enough for a better understanding of the implementation, so we have included them as Annexes.
2 Semantic Catalogue

In a nutshell, the main objective of the Semantic Catalogue is to provide an enhanced OGC CSW compliant [4] service that takes advantage of semantic annotations provided in the service descriptions. From a high level point of view we need therefore: (1) the services to be semantically annotated, (2) a semantic discovery algorithm that uses those semantic descriptions (as already defined in deliverable D5.3 [3]; details about the integration of the implementation can be found in Section 4.3) and (3) to integrate this algorithm into the standard OGC CSW operations for discovery.

2.1 Semantic service descriptions

Much of the effort of WP4 was to provide means to annotate semantically OGC service descriptions based on the Capabilities document, a standard document that each service has to provide describing metadata information about itself.

However, those descriptions cannot be directly stored in the catalogue (not only in the semantic version but on any standard OGC catalogue). The reason is that the catalogue, as defined by the OGC standard [7], has a broader use. It is aimed at storing not only services, but also any kind of geospatial resource. The information that is stored in the catalogue is called metadata and complies with different ISO standards (mainly ISO/TS 19139:2007 and ISO 19119:2005) [5] [6] that describe certain aspects of the resources that can be registered.

ENVISION describes OGC services based on a minimal service model called POSM [8] which can be used to describe semantically (using different ontologies and vocabularies) the nature and operation of the different type of services. This POSM models are integrated in the different description files that accompany an OGC service and can be retrieved from the standard OGC operator GetCapabilities.

In order to fully integrate our semantic approach in a standard OGC catalogue, we found a gap between the service descriptions that were semantically annotated and the ISO metadata that is used by the catalogue. Different catalogue implementations provide means of translating capabilities documents to ISO metadata, but it is based on the criteria of the implementer and there is no commonly accepted way of doing it. Besides, the semantic annotations would be lost in this translation because they are not mapped to any ISO property.

In summary, for this integration to be done, we need to consider the following requirements:

- ISO metadata is not linked to the semantic descriptions of the services. We need to provide some means for linking those two while preserving the compatibility with the standard.
- The discovery process should be self-contained, meaning that all the information needed for it should be easily accessible from the provided input. This implies that the POSM models should have pointers to all the information needed by the discovery, i.e., all the ontologies used in the description of the services have to be included in the model, so that the discovery can be carried out properly. In practice this means that the pointers to the descriptions should be dereferenceable for the discovery to work.

During the discovery process many different documents and descriptions are used. Figure 1 depicts the different connections that are involved in the semantic descriptions of the OGC services. Most of these models were developed as part of WP4.
Figure 1.- Relations between the descriptions involved in the semantic discovery.
For the semantic discovery, the most relevant documents are described next:

- **Metadata (OGC Service Capabilities):** this is the main file for every OGC Service. It has the description of the service from the OGC point of view. In order to annotate this document we use metadata definition capabilities of the OGC schema and add metadata pointers to both the WSDL file and the POSM model (the RDF model describing the service in a semantic way).

- **OGC ISO metadata:** this is the information that is stored in the OGC standard catalogue. As the catalogue only deals with ISO metadata information we have to provide means to relate the information published in the catalogue and the semantic description of the services. In our case we use the distribution information as defined in the standard to add two online resources, one pointing at the semantically annotated capabilities document and another one pointing at the resource information inside the Resource Module (the integration with this component will be explained in detail in section 3.5.2)

- **POSM:** the document containing the POSM model is the pillar of the semantic discovery. It contains the semantic description of the service, the conditions and the effects that will be used in the query containment algorithm. However, this model needs to carefully import all the needed ontologies, both the ontologies used in the model itself but also the ontologies used in the conditions and effects so that the query containment algorithm has all the ontological information needed to define if the services are a match or not.

In the following section we will explain all this dependencies with the example of the OGC service WPS.

### 2.1.1 WPS example

As we established in the previous point, the semantic discovery service needs several pieces being in place and accessible in order to perform the discovery algorithm. In this section we will present all these pieces using a WPS service as an example. As these documents tend to be very long, we will just show the parts that are directly relevant for the discovery.

#### 2.1.1.1 WPS GetCapabilities

POSM files are referenced from the metadata section of the process offerings (remember this example is a WPS). Note the use of arcrole, so that the links are labelled and can be easily automatically identified.

**Listing 1.- WPS GetCapabilities document.**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wps:Capabilities service="WPS" version="1.0.0" ... >
  <ProcessOfferings xmlns="http://www.opengis.net/wps/1.0.0">
    <ows:Identifier>Add</ows:Identifier>
    <ows:Metadata xlink:arcrole="http://role.org/posm">
      xlink:href="[POSMURL]/Add.rdf#Add"/>
    <ows:Metadata xlink:arcrole="http://role.org/wsd1">
      xlink:href="[WSDL URL]/Add.wsdl"/>
    <ows:Title>Add</ows:Title>
    <ows:Abstract>(a + b) through WPS</ows:Abstract>
  </wps:Process>
</wps:Capabilities>
```

Copyright © ENVISION Consortium 2009-2012
2.1.1.2 WPS ISO Metadata

The metadata description of the resource is independent from the Capabilities document. As we explained before, there is no standard way of transferring the information from the Capabilities to the ISO fields. The CSW catalogue implementations, as GeoNetwork [9], the one we use, define some mappings but other information is lost in the translation.

In the case of the semantic discovery, the information contained in the Capabilities is essential. As there is no semantic version of the ISO metadata format and we still need to provide this connection, we decided to add two descriptors to the ISO metadata. One for the semantically annotated capabilities document and another one for the resource definition of the service in the ENVISION platform. We used the field “description” to differentiate between them and the link type is always a HTTP link, i.e., an address that can be retrieved with a simple HTTP GET.

Listing 2. WPS ISO metadata.

```xml
<MD_Metadata ...>
  ...
  <identificationInfo>
    <SV_ServiceIdentification>
      ...
      <serviceType>
        <gco:LocalName codeSpace="www.w3c.org">OGC:WPS</gco:LocalName>
      </serviceType>
      <serviceTypeVersion>
        <gco:CharacterString>1.0.0</gco:CharacterString>
      </serviceTypeVersion>
    </SV_ServiceIdentification>
  </identificationInfo>
  <distributionInfo>
    <MD_Distribution>
      <transferOptions>
        <MD_DigitalTransferOptions>
          <onLine>
            <CI_OnlineResource>
              <linkage>
                <URL>
                  [Semantic Capabilities URL]/capabilities.xml
                </URL>
              </linkage>
            </CI_OnlineResource>
          </onLine>
        </MD_DigitalTransferOptions>
      </transferOptions>
    </MD_Distribution>
  </distributionInfo>
</MD_Metadata>
```
One important task in the process of providing the metadata is to define who and how it is created. This metadata is created in the publication phase of a service. The publisher has to guarantee that those fields are correctly provided. Inside the ENVISION platform this is responsibility of the Resource Module, the component which publishes semantically annotated services.

The discovery process needs to be aware of the connection between the metadata registered in the catalogue and the semantic descriptions. The links provided in the metadata will be then accessed by the discovery process in order to obtain the semantic descriptions that will be used as part of the discovery algorithm.
2.1.1.3 WPS POSM file

In the POSM document many parts are of high importance for the discovery:

- The POSM has to import all the needed ontologies. For that it should use the `owl:imports` construct and explicitly declare the ontologies used in the POSM itself, but also the ontologies used in the condition and effects parts.

- The conditions and effects have to be associated with the operations of the service, so that each operation might have different conditions and effects. This connection is done using a `sawsdl:modelReference`.

- The condition and effect class contain and `rdf:value` which defines the WSML axiom that applies to it. This axiom is represented with the human readable syntax of WSML.

Listing 3.- WPS process POSM.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF ... >
  <owl:Ontology>
    <owl:imports rdf:resource="http://purl.org/ifgi/wps#"/>
    <owl:imports rdf:resource="http://purl.org/ifgi/ows/0.2#"/>
    <owl:imports rdf:resource="http://www.other.org/ontology.rdf"/>
  </owl:Ontology>

  <posm:Service rdf:about="POSM URL/Add.rdf#Process_Add">
    <dc:title>Add</dc:title>
    <posm:hasOperation rdf:resource="http://purl.org/ifgi/ows/0.2#DefaultGetCapabilities"/>
    <posm:hasOperation rdf:resource="http://purl.org/ifgi/wps#DefaultDescribeProcess"/>
    <posm:hasOperation>
      <wps:Execute rdf:about="POSM URL/Add.rdf#Execute">
        <dc:title>Execute: Add</dc:title>
        <posm:hasInput rdf:resource="POSM URL/Add.rdf#ExecuteRequest"/>
        <posm:hasOutput rdf:resource="POSM URL/Add.rdf#ExecuteResponse"/>
        <sawsdl:modelReference rdf:resource="POSM URL/Add.rdf#EffectDefinition"/>
      </wps:Execute>
    </posm:hasOperation>
  </posm:Service>

  <wps:ExecuteRequest rdf:about="POSM URL/Add.rdf#ExecuteRequest">
    <dc:title>Execute Request: Add</dc:title>
  </wps:ExecuteRequest>
</rdf:RDF>
```
2.2 Semantic discovery and publication functionality

With the current release of the discovery service we fully integrate the semantic descriptions of the services in the discovery process. As it could be expected, adding full support for the semantic descriptions brought some architectural and operational changes to the prototype of the previous release.

In the following sections we describe, from the operational point of view, the functionality of the semantic discovery, divided into two main scenarios:

1. Publication of semantically annotated services.
2. Semantic discovery of services registered in the OGC catalogue.

We assume the ENVISION platform is used to publish semantically annotated services. Publication of other type of resources is possible in standard OGC CSW catalogues, but not in the scope of our work. For the publication of other types of resources, the underlying OGC Catalogue implementation should be used directly, in our case GeoNetwork [9].

Discovery, however, avoids stating that the services are semantically annotated. The discovery interface provides a standard CSW interface that could be used to discover non-semantically annotated services that are already registered in the catalogue. This is necessary because users of the ENVISION platform can also search for non-semantically annotated services and annotate them themselves.

In the following sections we clarify the steps carried out in each one of the process that are part of the semantic discovery. We assume familiarity with other components of the ENVISION platform, at least with the Resource Module and the WSML Goal Editor. More information about these components can be found in WP4 deliverables.

2.2.1 Publication of semantically annotated resources

The ENVISION platform allows users to semantically annotate services (previously added to the Resource Module). Once the service has been annotated it can be published in the catalogue.
In the ENVISION platform, the component that interacts with the publication interface is the Resource Module. This is why it is used as the initiator of the publication. However, as the Semantic Service discovery implements the CSW standard, any other compliant client could publish services, as long as the semantic metadata is properly provided.

Publishing a semantically annotated service in the catalogue requires certain steps, as shown in Figure 2 (we excluded the user from the activities, this will be explained along with the user interface in deliverable 2.3 [15]):

1. The Resource Module transforms the capabilities document of the service to be published in the correspondent ISO metadata, adding the links to the semantically annotated capabilities and its internal resource description document.

2. The Resource Module sends a CSW HTTP Transaction Insert request with the metadata to the semantic discovery service.

3. The semantic discovery service first redirects the transaction to the underlying CSW implementation (GeoNetwork in our platform) using the same standard message.

4. As a result, it will receive a UUID that identifies the newly added resource.

5. Once the service is registered in the catalogue, the semantic information is extracted following the links in the metadata information. All the models and ontologies are collected and stored in the reasoner’s knowledge base so that it can be accessed in the discovery phase.

6. After the semantic information is correctly stored, the UUID of the newly added resource is returned to the Resource Module.
2.2.2 Semantic discovery of services registered in the OGC catalogue

The main objective of the semantic discovery is providing users with a versatile, powerful and meaningful interface to discover services that might be relevant for them when creating environmental models.

OGC discovery is based on Filters. In order to support semantic descriptions (in our case WSML goals) in the queries, we design, following the rules of extension of the language defined in the standard, a new operator to be supported by catalogues that are semantically enabled. Our operator is called *matchesGoal* and its formal definition was introduced in D5.3 [3].

The purpose and main benefits of following this approach are:

- The operator is seamlessly integrated in the common OGC Filter schema.
- If a semantic query is sent to a catalogue that it is not semantically enhanced it will be just ignored, therefore, in a distributed configuration, many types of catalogues can cooperate with no integration effort.
- It is not easy to define WSML goals, so the simpler the mechanism the better. We leverage the creation of the goal to other components of the project and keep a simple readable way of representing them in the filters.
Figure 3.- Semantic discovery diagram

The initiation of the discovery process in the ENVISION platform could be done from different components. To show how the discovery works we will just use a generic discovery client.

Discovering a semantically annotated service in the catalogue requires certain steps, as shown in Figure 3 (we excluded the user from the activities, this will be explained along with the user interface):

1. The CSW HTTP endpoint of the discovery service receives a CSW HTTP GetRecords request with an OGC filter that contains the semantic operator.
2. The request is redirected to the underlying implementation of the standard OGC CSW catalogue (GeoNetwork).
3. This server will reply with a GetRecords response document with the results. These results are the services that match the non-semantic part of the OGC filter.
4. With this set of services, now the semantic properties have to be checked. The UUIDs are used to retrieve from the RDF repository the ontologies and descriptions of the services.
5. The set of ontologies and descriptions are sent to the reasoner, as well as the goal that was included in the filter.
6. The reasoner will check which services are matches, based on the objective goal and the semantic descriptions of the services, applying the query containment algorithm.
7. The identifiers of the matching services will be returned.
8. With these identifiers the discovery service can already prepare the final response with only the matching services and send it back to the caller.
3 Semantic Catalogue architecture

Figure 4 presents a detailed high-level view of the components involved in the Semantic catalogue architecture.

There are four main components: the discovery module provides the user interface, the semantic discovery service implements the endpoint and coordinates the rest of the components, the OGC Catalogue provides an operational OGC catalogue implementation and the reasoner deals with the semantic part of the discovery process.

Figure 4.- Semantic discovery architecture overview, second year release
3.1 Discovery Module (User Interface)

The user interface provides users with the appropriate mechanisms to define the query about the service that they are searching for. The novelties in the user interface are as follows (see Table 1):

- **Support for definition of WSML goals**, enabling the integration of the semantic discovery. The creation of the goal interface is taken from WP4, how this integration is done is detailed in section 3.5.1 WSML Goal editor.
- **Results from the discovery** are now presented in a different way, displaying more information about the resource.
- **All the components** have been redesigned and are now implemented with GWT (more details in the implementation section).

### Table 1.- Discovery module visual components implementation

<table>
<thead>
<tr>
<th>Visual Component</th>
<th>Type</th>
<th>Purpose</th>
<th>Y1 release</th>
<th>Y2 release</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keywords box</td>
<td>Input</td>
<td>Introduction of keywords related to the services to be discovered</td>
<td>YES</td>
<td>redesign</td>
</tr>
<tr>
<td>Spatial constraints</td>
<td>Input</td>
<td>Select the geographical area to be covered by the discovery</td>
<td>YES</td>
<td>redesign</td>
</tr>
<tr>
<td>Semantic query</td>
<td>Input</td>
<td>Define a WSML goal to be fulfilled by the services</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>Search button</td>
<td>Input</td>
<td>Trigger the discovery process</td>
<td>YES</td>
<td>redesign</td>
</tr>
<tr>
<td>Add to User collection button</td>
<td>Input</td>
<td>Trigger the process of sending the selected resources to the user collection.</td>
<td>YES</td>
<td>improved</td>
</tr>
<tr>
<td>Results area</td>
<td>Output</td>
<td>Display the discovery results</td>
<td>PARTIAL</td>
<td>YES</td>
</tr>
</tbody>
</table>

OGC Filter queries can contain different constructs according to the standard. We have selected the most commons ones as a significant example (keywords and geospatial boundaries).

3.2 Semantic Discovery Service

The Semantic discovery service itself provides a CSW HTTP standard interface. The difference with this component and any other standard OGC catalogue is that our service supports the semantic operator "matchesGoal" that we have defined. This implies that the service does special work in order to take advantage of the semantic annotations provided with the services, but adds no additional requirements to the catalogue infrastructure.
This component then acts as a controller for the semantic discovery while guaranteeing that the CSW HTTP binding standard is met. For all the operations not related with the semantic part of the discovery this component will rely on a standard implementation of an OGC catalogue. This means that those requests that do not explicitly involve a semantic processing will be just redirected to the underlying implementation that is linked to the semantic extension.

Table 2 summarizes all the functionalities supported by the CSW HTTP binding standard (which were already analysed in Deliverable 5.3).

<table>
<thead>
<tr>
<th>Request</th>
<th>HTTP method binding(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCapabilities</td>
<td>GET (POST)</td>
</tr>
<tr>
<td>DescribeRecord</td>
<td>POST(GET)</td>
</tr>
<tr>
<td>GetDomain</td>
<td>POST(GET)</td>
</tr>
<tr>
<td>GetRecords</td>
<td>POST(GET)</td>
</tr>
<tr>
<td>GetRecordById</td>
<td>GET (POST)</td>
</tr>
<tr>
<td>Harvest</td>
<td>POST(GET)</td>
</tr>
<tr>
<td>Transaction</td>
<td>POST</td>
</tr>
</tbody>
</table>

The operations that require semantic processing are the following:

- **GetRecords**: This is the main function used for discovery in the CSW standard. When this request is sent the semantic discovery process is triggered.

- **Transaction**: this request has three sub actions, Insert, Delete and Update and it is used to add new resources to the catalogue (what is also called publishing). When a new resource is published the semantic information has to be crawled and processed based on the annotations. After publication the semantic information will be ready to be used for the discovery.

- **Harvest**: harvesting is an advanced feature of the OGC catalogues but as the state of the implementations in the different CSW catalogues that we analysed is not complete or mature enough we have not included this operation in the scope of our catalogue.

### 3.3 Reasoner

The reasoner component provides the query containment algorithm used for the semantic discovery and also the needed infrastructure and libraries to be able to process the semantic annotations provided in the service descriptions. The reasoner component has also integrated an RDF repository (based on Sesame [10]) that will act as a semantic cache for the service descriptions and ontologies that need to be used during the discovery process.
3.3.1 RDF Repository

The rationale behind having a dedicated RDF repository for the service descriptions, instead of collecting them at discovery time, is twofold:

1. It seemed logical to have the semantic descriptions integrated in the catalogue. However, the catalogue implementations do not support this kind of information. As we have explained before, neither the CSW standard nor the ISO metadata information that is stored in the catalogue relate to semantic information. Storing the semantic information in the same place as the ISO metadata would require a custom catalogue implementation with its own storage system compatible with both types of descriptions. Our approach was nevertheless focused on providing the semantic capabilities as something “plugable” to any existing catalogue, so the semantic information is separated from the ISO metadata of the catalogue but linked by the UUID that every resource has.

2. As we anticipated in previous year release, retrieving the semantic descriptions every time the discovery process is launched is not efficient enough. This is why we added the RDF repository, a scalable data-base like system that holds the semantic information of the services stored in the catalogue.

This decision is also based on previous experience in other projects, like SWING [16], where the WSMX platform was based on simple file system storage. As we have moved to RDF representations we can now take advantage of the already available RDF repositories to carry out this functionality. This RDF repository can be easily queried by means of standard SPARQL queries too.

3.3.2 Semantic Reasoner

This component has just been adapted since the last release. It wraps the IRIS [11] datalog reasoner used for the query containment algorithm. Additionally it provides the necessary parsers and serialisers for the different ontologies and documents involved in the process, providing means to transform the WSML representations to RDF and viceversa.

3.4 OGC Catalogue

The catalogue server is the underlying implementation of a CSW catalogue that can be connected to the semantic discovery service. We are not dependent on any concrete implementation. We kept the communication based on the standard CSW HTTP binding, so any other implementation that complies with this standard is interchangeable. With this approach our semantic catalogue could use any CSW compliant catalogue, and likewise, every already existing CSW compliant catalogue could be extended with our semantic approach.

3.4.1 CSW access library

To facilitate the operation with the set of CSW and OGC standards we have a library that provides tools and utilities to ease this integration. The purpose of this library is to provide:

- An object model for the OGC schemas.
- A client library for the CSW HTTP binding.
- A standard set of serialisers and parsers for the different schemas used in the different CSW messages.
3.5 Integration with other components

The ENVISION platform is based on Portlets that, by definition, can collaborate and also be reused. In the semantic discovery service we do both. We use the WSML goal editor as defined in WP4 to provide the functionality of creating the goals of the discovery and also we connect to the Resource Module in order to add the discovered resources to the user collection for further usage.

3.5.1 WSML Goal editor

The WSML Goal editor is the component that enables the user to define a goal to be used for the discovery for the service, based on the ontologies previously loaded in the component.

This component is part of the WP4, so its description can be found in deliverable D4.4.

Figure 5.- WSML goal editor UI
3.5.2 Resource Module

The Resource Module is an important piece of the ENVISION platform because it manages the user collection of services, annotations and models. From the discovery point of view, the functionality of the Resource Module that has to be considered is basically that the Resource Module holds the semantically annotated services (done by the user) and enables the user to publish them in the semantic catalogue, for which the correspondent ISO metadata of the services has to be provided. The Resource Module also provides the functionality to add services that have been discovered to the user collection in order to annotate them and, regarding the semantic discovery, eventually publish them.

Figure 6.- Resource Module UI
4 Semantic Catalogue Implementation

For the ENVISION platform we decided to develop the client side components following the JSR 268 standard for Portlets. However, the Semantic discovery has additionally a special restriction; it has to comply with the OGC CSW HTTP binding, which makes it a server side component, not directly linked to any portlet. We have developed all the server side components as independent libraries. The implementation of the CSW HTTP binding is carried out by a simple Java Servlet that can be deployed in any Java compliant servlet container.

The software development process for the semantic discovery service has been done following a test-driven development. This methodology focuses on the functionality of the software, while leveraging the design to a refactoring process that is done continuously during the development. Comprehensive low-level diagrams or designs are not representative of the state of the software. We will however include basic class diagrams of the different components at the point of writing this deliverable.

The software lifecycle management is done with Maven. It is structured in three Maven modules:

- envision.discovery.csw: classes related to the OGC catalogue and utility classes.
- envision.discovery.reasoning: classes related to the reasoner and the RDF repository.
- envision.discovery: contains the core of the semantic catalogue: the user interface, theportlet description and the server side servlets.

For further understanding of the evolution and state of the implementation we refer to two of the tools used in our methodology, the issue tracker (http://kenai.com/jira/browse/ENVISION) where all the low level actions where registered and of course, the unit tests that are part of the each software component. These tests can be found in each component in the source code repository of the project (https://svn.kenai.com/svn/envision-portal). Both repositories have open read access due to the open source nature of the ENVISION project.
Figure 7.- Component diagram of the Semantic Catalogue
4.1 Discovery Module (User Interface)

The discovery module of the semantic catalogue (the user interface) has been developed using the Google Web Toolkit set of libraries [12]. GWT is a powerful platform for creating Web applications and interfaces in a Java-like fashion. The tools provided generate in the compilation phase the necessary javascript code needed in the client side of the application that can be deployed to any web server afterwards. We have chosen this platform because it facilitates the creation of web interfaces easily and relatively fast.

Listing 4.- JSON representation of a OGC Filter with the semantic operator

```
"ogc.Filter":
{
    "ogc.And":
    {
        "ogc.PropertyIsLike":
        {
            "$wildCard":"*",
            "$singleChar":"?",
            "$escapeChar":"\\",
            "ogc(PropertyName":"csw:AnyText",
            "ogc.literal":"*envision*",
        },
        "ogc.BBOX":
        {
            "ogc(PropertyName":":iso:BoundingBox",
            "gml.Envelope":
            {
                "gml.lowerCorner":"0.0 0.0","gml.upperCorner":"0.0 0.0"
            },
        },
        "[OTHER OGC PROPERTIES],
        "ogc.MatchesGoal":
        {
            "ogc.literal":"myGoal"
        }
    }
}
```

The user interface has the following components:

- **Discovery Portlet**: a JSR portlet that wraps the GWT functionality in order to be integrated in the ENVISION portal.

- **Discovery Module UI**: the set of components (textboxes, buttons, tables, etc) with which users can define the query, send it to the server, check the results and add them to their private collection if needed.

- **OGC Filter utils**: with the information present in the interface, an OGC filter has to be composed. This is done by some helper classes that represent the OGC Filter model in the client side. This filter can be then easily serialized to JSON.

- **OGC Records utils**: the response from the server has to be processed by the interface and shown to the user. We have shown them in a tabular fashion and display some checkboxes so that services can be added to the users collection (via the Resource Module).
For the implementation of the interface we decided to add an additional content negotiation option to the Semantic CSW HTTP interface. The OGC standards work with XML based documents, but for web clients it is also common to provide JSON representations of the objects returned by the server. In our case, the content of the replies of the server can be negotiated by the use of HTTP Accept header. If the header is "application/xml" the server will reply with the xml version. If the header is "application/json" the server will reply with the JSON serialization of the XML original reply.

**Listing 5.- XML representation of an OGC Filter with the semantic operator**

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<csw:GetRecords ...>
<csw:Query typeNames="gmd:MD_Metadata">
<csw:ElementSetName>full</csw:ElementSetName>
<csw:Constraint version="1.1.0">
<ogc:Filter>
<ogc:And>
<ogc:PropertyIsLike wildCard="*" singleChar="?" escapeChar="\">
<ogc:PropertyName>csw:AnyText</ogc:PropertyName>
<ogc:Literal>Fire</ogc:Literal>
</ogc:PropertyIsLike>
<ogc:PropertyIsLike>
<ogc:PropertyName>iso:BoundingBox</ogc:PropertyName>
<gml:Envelope>
<gml:lowerCorner>-34.6 51.1</gml:lowerCorner>
<gml:upperCorner>-17.3 38.2</gml:upperCorner>
</gml:Envelope>
</ogc:PropertyIsLike>
<ogc:BBOX>
<ogc:PropertyName>iso:BoundingBox</ogc:PropertyName>
</ogc:BBOX>
[OTHER OGC Properties]
<ogc:MatchesGoal>
<ogc:Literal>WSML GOAL</ogc:Literal>
</ogc:MatchesGoal>
</ogc:And>
</ogc:Filter>
</csw:Constraint>
</csw:GetRecords>
```
The Semantic Discovery service provides the CSW HTTP interface and coordinates the semantics related operations. The implementation is based on a HTTP servlet and has the following packages and classes (see Figure 9):

- **impl**: contains the HTTP endpoint and the implementation of the Semantic Discovery Service, the class that coordinates the semantic discovery.
- **csw**: every operation from the CSW has a correspondent wrapping action that triggers the internal process needed to fulfill the request (command pattern).
- **semantics**: based on the services metadata, the semantic service descriptions must be retrieved from their original addresses. These set of classes provide the functionality to identify which files to retrieve and which information to extract.

![Discovery Module UI](image-url)

**Figure 8.- Discovery Module UI.**

### 4.2 Semantic Discovery service

The Semantic Discovery service provides the CSW HTTP interface and coordinates the semantics related operations.

The implementation is based on a HTTP servlet and has the following packages and classes (see Figure 9):

- **impl**: contains the HTTP endpoint and the implementation of the Semantic Discovery Service, the class that coordinates the semantic discovery.
- **csw**: every operation from the CSW has a correspondent wrapping action that triggers the internal process needed to fulfill the request (command pattern).
- **semantics**: based on the services metadata, the semantic service descriptions must be retrieved from their original addresses. These set of classes provide the functionality to identify which files to retrieve and which information to extract.
• **utils**: this package provide generic URL retrieval facilities to be used by the rest of the classes. It also defines a BasicAuthenticator class, used in cases where the URLs need authentication (in our case, the Resource Module and the GeoNetwork server).

![Semantic Discovery Service Class Diagram](image.png)

**Figure 9.- Semantic discovery service class diagram**

The semantic discovery service also provides HTTP based content negotiation. Clients can retrieve the results either in the standard XML format or in JSON format. The JSON format is automatically generated by means of the open source jettison library [13].
4.3 Reasoner

The reasoner component provides the functionality associated to the query containment algorithm. The underlying reasoner used is IRIS 0.8.1 [11] which was already updated in year 1 release to support Query Containment.

The reasoner also provides a facade for an RDF repository where the service descriptions can be stored in RDF format.

4.3.1 RDF Repository

For the RDF repository, we have deployed a Sesame Server with an HTTP connection used for storing and accessing the service descriptions (in RDF). This server has a context called Envision and the description of each service is stored as a NamedGraph with the UUID provided by the catalogue, acting as the link to the semantic information stored in the RDF repository, which also contains the ontologies defined in the POSM files that have been previously retrieved and loaded into the repository.
4.3.2 Semantic Reasoner

The Semantic Reasoner has the responsibility to find services that fulfil the user’s goal as defined in the queries. In our approach this is done using a query containment algorithm based on the semantic descriptions of the services, basically by checking if the goal presented by the user is logically implied by the semantic descriptions of the services. The Semantic Reasoner component provides the access to this functionality by means of a simple API.

The needed information for this process is defined next:

- **Goal**: the goal is defined in the OGC query using the `matchesGoal` operator.
- **Effects of the services to check**: this information is stored in the RDF repository the moment a service is published, so it can be retrieved later with a simple SPARQL query.
- **Ontologies**: The ontologies used in the service descriptions are needed to check the query containment. These ontologies are also extracted from the imports in the POSM files and stored in the RDF repository. When needed they can be queried and sent to the IRIS reasoner.

4.4 OGC Catalogue

The OGC Catalogue is the component that provides an implementation of the functionality of the standard CSW Catalogue, without the semantic discovery enhancement. Any catalogue that has a CSW HTTP binding implementation is compatible with our semantic extension. Basically the functionality of the catalogue that we assume is fully functional is:

- CSW HTTP binding.
- Support for storage of ISO metadata information of services (and more general of resources).
- OGC Filter query support based on the standard operators (semantics not needed).
- ISO metadata editor (we do not support the editing of metadata).

4.4.1 GeoNetwork

There exist already different CSW compliant catalogues as open source solutions (see Deliverable 5.1). From the set of available catalogues, we selected one, GeoNetwork [9], to be used as the underlying infrastructure.

GeoNetwork is, from our point of view, a significantly complete solution that is also under active development. For the first year release we deployed version 2.6.3 of the server but for the new release we have updated to the 2.7.0 snapshot release. This is still a version under development, but there are some new features that we considered interesting:

- Improved native support for all types of OGC services information metadata.
- Improved metadata editor interface.

Besides this updated features we had also to adapt the implementation of GeoNetwork itself. In the publication process we detected that the server was not providing the reply using the appropriate namespace, so that the client could not properly parse the response. In Listing 6 we can see the right XML response to the TRANSACTION INSERT operation. The namespace of the identifier (which is vital for us to link the ISO metadata with the semantic descriptions) was not defined, so we created a patch for GeoNetwork that solves our problem.
4.4.2 CSW access library

The set of different standards and schemas used by OGC is overwhelming. In order to provide an easier access to the different types of messages and documents that can be used in the CSW HTTP binding of the OGC Catalogue we have integrated a open source library that has almost all the object models and schemas for the OGC related standards.

This library is part of an open source project called Geotk (http://www.geotoolkit.org/). Geotk is the reference implementation of GeoAPI 3.0 interfaces. It provides data structures for geographic data and associated metadata along with methods to manipulate those data structures. The library rests on a set of utility classes which include a framework for factory classes. The Geotk metadata module forms the base of the library and enables the creation of metadata objects which comply with the ISO 19115 metadata model and which can be read from or written to ISO 19139 compliant XML documents.

Using this library we provide three main functionalities:

- Support for the semantic operator matchesGoal as part of the object models defined for the OGC Filter standard.
- Generic CSW HTTP client.
- Helper classes to extract from the documents (using XPath) Envision specific information from the documents (like the pointers to the semantic descriptions).

Listing 6.- Transaction Insert response example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<csw:TransactionResponse xmlns:csw="http://www.opengis.net/cat/csw/2.0.2">
  <csw:TransactionSummary>
    <csw:totalInserted>1</csw:totalInserted>
    <csw:totalUpdated>0</csw:totalUpdated>
    <csw:totalDeleted>0</csw:totalDeleted>
  </csw:TransactionSummary>
  <csw:InsertResult handleRef="handleRefValue">
    <csw:BriefRecord>
      <dc:identifier xmlns:dc="http://purl.org/dc/elements/1.1/" cd6c59e0-01e0-4e02-8805-b7c223d24e1c>
      </dc:identifier>
    </csw:BriefRecord>
  </csw:InsertResult>
</csw:TransactionResponse>
```
4.5 Integration with other components

The Semantic Catalogue does not work independently, it is connected to several components from the ENVISION platform.

4.5.1 WSML Goal Editor

At the portlet level (mainly user interface) the discovery module relies on the WSML Goal Editor to provide the WSML definition of the goal.

The communication between both Portlets is done on the client side via Javascript. The Discovery Portlet provides a function to update the WSML goal so that the definition of the goal can be displayed in the discovery interface and sent to the catalogue as part of the normal query.

More details about the inter-portlet communication using Javascript can be found in the Annex.
4.5.2 Resource Module

The Resource Module is a component inside the ENVISION platform that manages, among other things, the services and annotations of a particular user collection. The interaction with the Resource Module is bidirectional but is using different interfaces:

- When the user wants to publish a semantically annotated service, the Resource Module has to deal with the process of publication in the OGC CSW Catalogue. As the catalogue implements the CSW HTTP binding, this communication can be done in a standard way directly to the server. There is no need to add additional client-side communication between the portlets.

- When users discover services and want to add them to their resource collection (i.e. imported to the Resource Module), this action is triggered from the Discovery Module portlet. In this case, the communication is done using the Resource Module API provided by the Resource Module and after the import function is done, the user interface is notified by means of a Javascript update function.
5 Conclusions

In this deliverable we presented the second year release of the semantic discovery engine and integration with an OGC open source catalogue. We have extended the previous version and added for this release full support for semantic queries based on WSML goals and query containment algorithms, as well as providing the needed software architecture.

We have now an independent component that can be coupled with any OGC CSW HTTP compliant catalogue to make it compatible with the semantic operator. We have also provided a user interface for the discovery inside the ENVISION platform, wrapped in a portlet and provided the necessary interface to connect the portlet with both the Resource Module and the WSML goal editor in order to fully integrate the functionality provided by the ENVISION platform.
Annex

Wrapping a GWT interface in a portlet

Once the interface and the functionality is ready it has to be wrapped into the platform component model, the portlets. There are some alternatives when doing this integration, based on the version and implementation of the portlet container used. In our case, and for the GWT version 2.4.0 and the portlet container Liferay 6.0.6 the process was as follows (we assume some knowledge about the platforms we are referring to):

1. Create a GenericPortlet that will be the main class of our wrapper. Define a simple view with an associated JSP file.

```java
public class DiscoveryPortlet extends GenericPortlet {
    private static final String NORMAL_VIEW = "/jsp/discovery.jsp";
    private PortletRequestDispatcher normalView;

    public void init(PortletConfig config) throws PortletException {
        super.init(config);
        normalView = config.getPortletContext().getRequestDispatcher(NORMAL_VIEW);
    }

    public void doView(RenderRequest request, RenderResponse response)
    throws PortletException, IOException {
        normalView.include(request, response);
    }

    public void destroy() {
        normalView = null;
        super.destroy();
    }
}
```

2. In the JSP file just create a script tag pointing at the javascript file generated from the GWT compilation (based on the compilation strategy and naming used). Also define an empty div with a unique identifier inside your project. This identifier is the one you need to use in the module when creating the user interface.

```html
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
<portlet:defineObjects/>

<script src="<%=request.getContextPath()%>/discoverymoduleui/discoverymoduleui.nocache.js">
</script>

<div id="discovery.portlet"/>
```
3. Provide the portlet.xml file descriptor for the portlet container using the simple portlet defined in step 1.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"/>
<portlet>
    <portlet-name>discovery-portlet</portlet-name>
    <display-name>discovery-portlet</display-name>
    <portlet-class>
        at.sti2.envision.discovery.portlet.DiscoveryPortlet
    </portlet-class>
    <expiration-cache>0</expiration-cache>
    <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
    </supports>
    <portlet-info>
        <title>Discovery Portlet</title>
        <short-title>DiscoveryPortlet</short-title>
        <keywords>envision, discovery</keywords>
    </portlet-info>
</portlet>
</portlet-app>
```

4. Deploy the JSP accordingly to the servlet container.

In our case we leverage the whole compilation and deployment system to Maven so we just need to make the adjustments explained before.

**Client-Side Inter-Portlet Communication with simple Javascript**

As a guided example of inter-portlet communication based on Javascript, as part of the work of this deliverable the following guide was produced, which can also be found at the project’s WiKi (http://kenai.com/projects/envision/pages/GuideJavascriptEvents).

This guide shows how two portlets can exchange information using mere javascript functions. This is a "quick-and-dirt" way of communication, for a more standard solution you should use Public Render parameters or Events. Some description of these methods and the correspondent examples can be found here [http://blog.xebia.com/2009/04/inter-portlet-coordination-with-jsr-286-JSR-286-Interportlet-Communication](http://blog.xebia.com/2009/04/inter-portlet-coordination-with-jsr-286-JSR-286-Interportlet-Communication).

For the example we will define two portlets, a Sender and a Receiver. The Sender will be able to call a javascript function defined by the Receiver, which will update a label defined in the presentation view.

To use this type of communication, we just need to include the Javascript functions and calls in the JSP view of the portlet.
Listing 7.- Step 1: Define the Javascript function

```javascript
1. <script type="text/javascript">
2.     var Receiver = {};
3.     Receiver.displayMessage = function(sender, message) {
4.         document.getElementById("text").innerHTML += "Received message: "
5.             + message + " from: <" + sender + ">
6.     );
7. </script>
```

- **Line 2:** We define an object that will act as an artificial "namespace". This is not mandatory but recommended for clarity.
- **Line 3:** We define a function inside the namespace object. In this example the function receives two parameters, the sender of the message and the message itself.
- **Line 4-5:** What the function does is up to the developer. In this example it just updates a label in the view of the portlet that contains the id of the sender and the message passed.

Listing 8.- Step 2 Invoking the Javascript function from other portlet

```html
<button type="button"
    onclick="Receiver.displayMessage('<portlet:namespace/>','hello');">
    Send hello message - javascript!</button>
```

Once the Receiver portlet is set up, we already can invoke the function from any other portlet present in the same page of the portal. For that we use the namespace object we have defined and the correspondent function signature.

We have defined a button which defines the 'onclick' event as a call to the javascript function with the portlet namespace and the message "hello"

Remarks:

- This is an easy way to communicate two portlets by sharing javascript functions.
- Depending on the desired functionality there might be problems if many instances of the portlet are active in the same page.

Source code:

The source code of the two portlets that communicate using javascript function and also a simple standard portlet event can be downloaded here:

- [Receiver Portlet](https://svn.kenai.com/svn/envision~portal/portlets/examples/receiver-portlet)
- [Sender Portlet](https://svn.kenai.com/svn/envision~portal/portlets/examples/sender-portlet)
References

[3] Larizgoitia, I; Toma, I; D5.3 Integration of the open source catalogue with the semantic discovery engine – Version 1, ENVISION, 2010.
[14] Larizgoitia, I; Toma, I; D5.3 Integration of the open source catalogue with the semantic discovery engine – Version 1, ENVISION, 2010.
[15] Florian Husson (BRGM), François Tertre (BRGM), Roy Gronmo (SINTEF), Iker Larizgoitia (UIBK), Patrick Maué (UoM), Henry Michels (UoM), Miha Grčar(JSI); D2.3 Environmental Semantic Web Portal Version 2 – ENVISION, 2011

Copyright © ENVISION Consortium 2009-2012
**Acronyms**

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OGC</td>
<td>Open Geospatial Services</td>
</tr>
<tr>
<td>WSML</td>
<td>Web Services modelling language</td>
</tr>
<tr>
<td>CSW</td>
<td>Catalogue Services for the Web</td>
</tr>
<tr>
<td>POSM</td>
<td>Process Oriented Service Model</td>
</tr>
<tr>
<td>WPS</td>
<td>Web Processing Service</td>
</tr>
<tr>
<td>UUID</td>
<td>Universally Unique Identifier</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>GWT</td>
<td>Google Web Toolkit</td>
</tr>
</tbody>
</table>